# basic compiler for trs-80 model 1

Microsoft Consumer Products
400 - 108th Ave. NE, Suite 200
Bellevue, WA 98004

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

## CONTENTS OF THE BASIC COMPILER PACKAGE

The BASIC Compiler package contains:

Two diskettes
A binder with Two manuals

## DISKETTES

The two diskettes in your BASIC Compiler package contain program files. The arrangement of the files is:

Diskette #1 - BASCOM/CMD - (compiler program)
L80/CMD - (linking loader)
GRAF/BAS - (sample program)

Diskette #2 - BASLIB/REL - (runtime library)
BRUN/CMD - (runtime module)

Both diskettes contain TRSDOS and Disk BASIC.

IMPORTANT

> You should make backup copies of these diskettes immediately. Store the masters in a safe place and work with the backup copies. Use the TRSDOS BACKUP command to make your copies.

## MANUALS

The two manuals in the BASIC Compiler package are:

BASIC Compiler User's Manual (this manual)
BASIC-80 Reference Manual

## BASIC COMPILER USER'S MANUAL

### PURPOSE

This manual is designed for users who are unfamiliar with the compiler as a programming tool. Therefore, the manual provides a step by step introduction and guide to BASIC Compiler and its use. At the same time, this manual assumes that the user has a working knowledge of the BASIC language. In the areas which derive from BASIC programming, this manual treats the material in quick summary. If you need additional assistance with BASIC programming, refer to the RESOURCES FOR LEARNING BASIC Section below for references.

### ORGANIZATION

This manual is organized to give you a gradual and working introduction and explanation of BASIC Compiler.

Chapter 1, Introduction, provides brief descriptions of the contents of the BASIC Compiler package, plus general information about system requirements, royalty information, and resources for learning BASIC programming. We have also included a word about Microsoft Consumer Products.

Chapter 2, Procedures, takes you step by step through the process of compiling; from editing to compiling to link loading to running, using a sample program to provide you an working introduction to the workings of BASIC Compiler.

IMPORTANT

> We recommend that you use the sample program before compiling your own programs. Then, read carefully Chapter 3, Section 3.1 EDITING, before attempting to compile your existing programs. You may need to edit your existing programs before they will compile successfully.

Chapter 3, Technical Descriptions, describes the workings of BASIC Compiler. You will find descriptions of command line syntax; what each part of a command line does and the variations you can make in your commands to the BASIC Compiler processes. You will find the error messages in this chapter, too.

At the end of the manual is an Index to help you locate topics for reference and review.

## NOTATION

When statements or commands are described in this manual, certain types of notation are used to indicate (1) what words and punctuation must be entered exactly as shown, (2) what entries must be typed in but the programmer selects the specific word(s), and (3) what words and punctuation are optional.

The notation used in this manual is:

1. Portions of statements and commands shown in CAPITAL LETTERS must be entered as shown.

2. Portions of statements and commands shown in lower case and surrounded by angle brackets (<>) indicate entries selected by the programmer. Lower case items in angle brackets are usually <filename>s.

3. Portions of statements and commands enclosed in square brackets ([]) indicate that the entries are optional; the programmer may include or omit these entries.

4. Ellipses (...) are used to indicate that any number of similar entries may be entered, up to the length of a line.

5. Braces ({}) indicate that the enclosed entries offer a choice; choose one of the entries enclosed in the braces.

6. Punctuation must be included where shown. Spaces are considered punctuation in this sense. However, angle brackets, square brackets, and braces are omitted from this rule unless specifically noted otherwise.

## BASIC-80 REFERENCE MANUAL

The BASIC-80 Reference Manual describes the syntax and usage of the latest version of Microsoft's standard BASIC interpreter. BASIC Compiler supports the BASIC-80 interpreter along with TRS-80 LEVEL II BASIC and TRS-80 Disk BASIC. Manuals for the two TRS-80 BASIC interpreters were supplied to you when you bought your system. The BASIC-80 Reference manual is supplied to you with BASIC Compiler so that you have a guide to the statements and commands that BASIC Compiler supports but which are not a part of LEVEL II BASIC or Disk BASIC. Note, however, that the BASIC-80 software is <u>not</u> supplied in the BASIC Compiler package, but is a separate product.

BASIC Compiler supports the statements and commands of all three of the BASICs mentioned except the following commands:

| | | | | | |
|---|---|---|---|---|---|
| AUTO | CLEAR | CLOAD | CSAVE | CONT | DELETE |
| EDIT | LIST | LLIST | RENUM | SAVE | LOAD |
| MERGE | NEW | COMMON | IN#-1 | OUT#-1 | SYSTEM |

IMPORTANT

> Language, operational, and feature differences between BASIC Compiler and the BASIC interpreters, described in Chapter 3, Section 3,1 EDITING, are charted in Table 3.0 at the beginning of Chapter 3. If you plan to compile a BASIC program you have written already, first review the information in Table 3.0 and the sections under EDITING, then make any changes necessary.

SYSTEM REQUIREMENTS

Microsoft BASIC Compiler can be used with the Radio Shack TRS-80 Model I computer with a minimum of 48K RAM and one disk drive. We recommend two drives, however, for easier operation. The most efficient and convenient operation of your TRS-80 Model I with BASIC Compiler can be attained when three drives are available.

## RESOURCES FOR LEARNING BASIC

Microsoft provides complete instructions for using BASIC Compiler. However, no teaching material for BASIC programming has been supplied. The BASIC-80 Reference Manual included in the BASCOM package is strictly a syntax and semantics reference for the Microsoft BASIC-80 language.

If you are new to BASIC and need help learning to program in this language, we suggest the following references:

1. Albrecht, Robert L., LeRoy Finkel, and Jerry Brown. BASIC. John Wiley & Sons, 1973.

2. Simon, David E. BASIC from the Ground Up. Hayden, 1978.

3. Dwyer, Thomas A. and Margot Critchfield. BASIC and the Personal Computer. Addison-Wesley, 1978.

4. LEVEL II BASIC Reference Manual. Fort Worth: Radio Shack, A Division of Tandy Corporation, 1979.

5. TRSDOS & DISK BASIC Reference Manual. Fort Worth: Radio Shack, A Division of Tandy Corporation, 1979.

# CHAPTER 2

# PROCEDURES

## NOTE

> Before beginning these procedures, make backup copies of both BASIC Compiler diskettes, using the TRSDOS BACKUP command. Both backup copies should be write protected to prevent TRSDOS writing to them. Then store your master diskettes in a safe place and work with the backup copies.

This chapter provides step by step instructions for using BASIC Compiler.

The steps are outlined using a sample program so that you can learn to use BASIC Compiler by doing.

We strongly recommend that you compile the sample program before compiling your own programs because the sample will compile with no problems. Before compiling your own programs, read Chapter 3 thoroughly. It contains important information that may affect compilation.

The best way to become familiar with BASIC Compiler is through the experience of using it. If you enter the program and commands exactly as described in this chapter, you should have a successful session with BASIC Compiler. If a problem does arise, check each step and redo the procedures very carefully. As you enter each line and command, check your entries for accuracy.

The four procedures for using BASIC Compiler are:

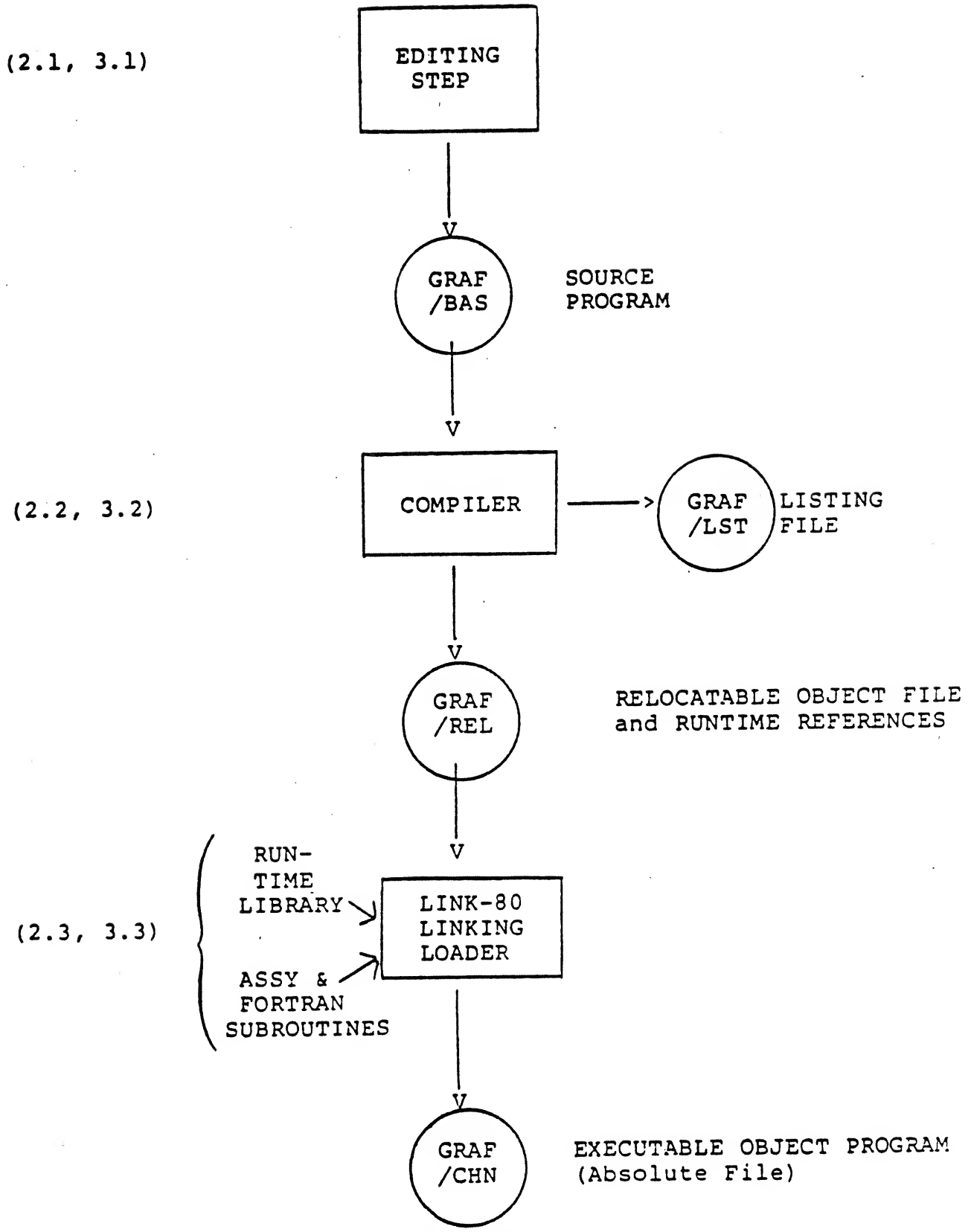
1. Editing (entering and correcting the BASIC program) Since this is the procedure you have been using to program in BASIC on your TRS-80, we have included the sample program GRAF/BAS on diskette #1.

2. Compiling (converting the BASIC program to object code)

3. Link Loading (converting the object code to executable code)

4. Running (executing the program)

Refer to the diagram on the next page to see how these processes relate.




NOTE: Numbers in parenthesis refer to applicable text sections

Figure 2.0: BASIC Compiling Processes

## 2.1 EDITING

The process of Editing involves entering the program, statement by statement, then editing to eliminate bugs, such as syntax errors, typographical errors, and erroneous data. The BASIC interpreter is used for the Editing procedures.

The basic steps are:

1. Insert a TRSDOS diskette into disk drive 0.

   A TRSDOS diskette contains Disk BASIC. TRSDOS and Disk BASIC are included on both BASIC Compiler diskettes. Both are also supplied with your Model I.

2. Power up your TRS-80

   When you turn on the power, TRSDOS will be booted automatically. The screen will display the messages:

   TRSDOS - DISK OPERATING SYSTEM - VERSION X.X

   DOS READY

   _

   The underline is the TRSDOS command level prompt.

3. Load Disk BASIC

   Enter the command

   BASIC

   (As usual when entering a BASIC program, press <ENTER> at the end of each line.)

   Disk BASIC will be loaded and respond with the message "HOW MANY FILES?"

   Press <ENTER>.

   Disk BASIC will respond with the message "MEMORY SIZE?"

   Press <ENTER>.

Disk BASIC will respond with the message

```
RADIO SHACK DISK BASIC VERSION X.XX
READY
>
```

4. Create a Disk BASIC program.

Enter a BASIC program as you normally do with Disk BASIC.

We assume that you are familiar with this process. Therefore, to save you the time needed to type in the program, we have included on diskette #1 a BASIC file called GRAF/BAS, which is the sample program listed below.

You should load GRAF/BAS now and list it to be sure that the disk file matches the listing given below. If you want to save GRAF/BAS to another diskette before continuing, read step 6 below first (save the program with SAVE "GRAF/BAS",A).

NOTE: The following program is a demonstration program we have included to give you a practical example of code generated by BASIC Compiler. You will find that it also gives you some interesting output. This sample program graphs a random selection of numbers between 0 and 127.

SAMPLE BASIC PROGRAM GRAF/BAS

```
100  DEFINT A-Z:DIM C(128)
200  CLS
300  PRINT @0, "RANDOM NUMBERS, UNIFORM = 1, NORMAL = 6 (NEG
NO. TO STOP)"
400  INPUT "SELECTION";N:IF N<1 THEN END
500  CLS:PRINT @20, "HIT ANY KEY FOR MENU"
600  FOR I = 0 TO 127
700  C(I) = 48
800  NEXT I
900  A$ = INKEY$:IF A$ <> "" THEN 200
1000 SUM = 0
1100 FOR J = 1 TO N
1200 SUM = SUM + RND(128)-1
1300 NEXT J
1400 RN = SUM/N
1500 C(RN) = C(RN)-1
1600 IF C(RN) < 0 THEN 200
1700 SET (RN,C(RN))
1800 GOTO 900
```

5. Check for errors in your source program.

At this point, it is a good idea to check the program for syntax errors because it is much easier to correct errors before compilation than afterwards. You should find no errors in GRAF/BAS. But, when you compile your own programs, you will find this step very useful. You cannot edit a compiled file. So if errors are discovered during or after compilation, you will have to return to the Disk BASIC interpreter to edit the file, then return to BASIC Compiler to recompile.

The fastest syntax check is simply to run your program.

Enter RUN

If any syntax errors exist in the program, Disk BASIC will display the message SYNTAX ERROR followed by the line number.

```
SYNTAX ERROR IN ### (line number)
READY
### _ (line number)
```

At this point, you will be in edit mode. Edit the line as usual with Disk BASIC, then enter RUN again. If no other syntax errors are displayed, your program is ready to be tested with the compiler.

(Even though you can assume that GRAF/BAS contains no syntax errors, running it will give you an idea of the speed programs run under the interpreter.)

IMPORTANT: Even though a program has no syntax errors under the interpreter, it may still have syntax errors under the compiler. Note also that BASIC programs with statements and commands from BASIC-80 but not in Disk BASIC will give interpreter errors but not compiler errors. Refer to Table 3.0 at the beginning of Chapter 3 and to Section 3.1 EDITING, for more information.

6. Save the program on diskette.

BASIC Compiler performs compilations of disk files. Therefore, any BASIC source program, must be saved on diskette before you can compile it. The GRAF/BAS program should now be

saved onto a program diskette. The program diskette may be any diskette you choose. You will want to save GRAF/BAS onto a different diskette than you loaded it from, since you loaded it from BASIC Compiler diskette #1. For now, use a diskette with a copy of TRSDOS on it.

To save your source program in ASCII format on this diskette, use the BASIC interpreter's SAVE command:

ACTION # 1.

```
SAVE "GRAF/BAS:d",A
```

GRAF/BAS is the filename we are giving this program. We are using the /BAS filename extension because BASIC Compiler recognizes programs with the /BAS extension by default.

:d represents the drive number where you have inserted the diskette. If the drive number is 0, or if you have only one drive, the :d specification may be omitted.

IMPORTANT

> The ",A" in the SAVE command line tells Disk BASIC to save the file in ASCII format. If the ",A" is omitted, the file is saved in its usual compressed format. However, because BASIC Compiler compiles ASCII formatted files only, you must always include the ",A" in the SAVE command line.

BASIC will respond with the message

```
READY
>
```

There is now a BASIC program called GRAF/BAS on your diskette that is ready to be compiled.

7. Return to TRSDOS by entering CMD"S".

## 2.2 COMPILING

Now that you have saved your BASIC program in ASCII format with the filename extension /BAS, you are ready for the compiling procedures.

NOTE

If you saved your source program without /BAS at the end of the filename, when you try to compile the program following the examples in this chapter, you will receive the error message

```
*** ERRCOD=24, FILE NOT IN DIRECTORY ***
     <FILE=GRAF/BAS/EEE:D>
   REFERENCED AT X'4BFB'
```

To eliminate this error, use the TRSDOS RENAME command to rename the program with the /BAS extension.

1. Perform a Compiler syntax check.

   The last step in the Editing procedure is also the first step in Compiling.

   To check for compiler syntax errors, you are actually going to compile the program, but you will not create a file for the compiled code. Rather, you are simply directing the compiler to display any error messages on the screen.

   From this point on, the procedure varies for one-drive and multi-drive systems. The basic difference is, in most cases, that with one-drive systems you must switch diskettes often while with a multi-drive system there is little switching of diskettes. For the steps where differences between one-drive and multi-drive systems apply, the steps are shown in side by side columns. Select the column appropriate to your configuration. Steps given in single columns apply to all configurations.

NOTE

If you have more than one drive, you have several choices in the arrangement of diskettes in the drives. If you have three or more drives, you can put the diskettes in any drive and simply leave them there. You need only remember that any save command or switch which saves a file must include the drive number as part of the filename.

If you have two drives, we suggest you place the BASIC Compiler diskettes in drive 0 (swapping them as necessary). This allows you to use a formatted diskette without TRSDOS for your files. (Both BASIC Compiler diskettes contain TRSDOS.) The result is more disk space for your programs.

A second choice for two drive systems is to copy the four BASIC Compiler modules onto one diskette (without TRSDOS). The result of this method is that you no longer need to swap diskettes #1 and #2. However, under this method your program diskette must contain TRSDOS and must be in drive 0, while the diskette with the four BASIC Compiler modules must be in drive 1.

ACTION # 2

**ONE-DRIVE SYSTEM**

To perform the syntax check on the source file, remove the diskette containing GRAF/BAS from the disk drive.

Insert Diskette #1 (the one which contains BASCOM).

Enter BASCOM

BASCOM responds with:
*

Remove BASCOM diskette #1 and insert the diskette which contains GRAF/BAS.

Enter =GRAF

These commands load BASIC Compiler and compile the source file without producing an object file or listing file.

**MULTI-DRIVE SYSTEM**

To perform the syntax check on the source file, place Diskette 1 (which contains BASCOM) in drive 0, inserting the diskette which contains GRAF/BAS in drive 1.

Then enter:

BASCOM =GRAF:1

This command loads BASIC Compiler and compiles the source file without producing an object file or listing file.

NOTE: Most usual call is:

BASCOM =name:1-E

where name is the program filename to be compiled.

If you want to stop the compiling process, and you have entered only BASCOM and nothing more, you can exit to TRSDOS by pressing the <BREAK> key. Otherwise, you have to reboot using the <RESET> key.

2. Watch for error messages.

If you have made any syntax errors, a two-letter code will appear on the screen with an arrow pointing to the source of the error. (See Chapter 3, Basic Compiler ERROR MESSAGES under Section 3.2 BASIC COMPILER, for definitions of the codes.) You should not see any error codes with GRAF/BAS.

NOTE

The error codes are displayed much too fast to read. If there is no more than one screenful of error codes, this is no problem. However, if there is more than one screenful, you must generate a listing file in order to read the codes. How to generate a listing file is covered in step 5 below.

3. Note the message on the screen.

After you have entered a command line and BASCOM has finished compiling, program control returns to TRSDOS, displaying the message:

xxxxx FATAL ERROR(S)

xxxxx BYTES FREE

DOS READY

4. Correct any fatal errors.

If you get error messages, return to Disk BASIC and correct the errors. You should get no errors with GRAF/BAS, but you might get errors when you compile your own programs. Remember: Some statements and commands are treated differently by BASIC Compiler than by the interpreter. See Chapter 3, Section 3.1 EDITING, for descriptions of these differences.

If no errors were encountered, you are ready to go on to the next set of procedures: compiling a relocatable object file.

5. Enter the command line.

Follow the steps for your configuration.

ACTION # 3

**ONE-DRIVE SYSTEM**

Insert BASCOM diskette #1 in the drive.

Enter BASCOM

When the asterisk (*) prompt appears, remove BASCOM diskette #1 from the drive.

Insert the diskette which contains GRAF/BAS into the drive.

Enter GRAF,GRAF=GRAF

**MULTI-DRIVE SYSTEM**

Insert BASCOM diskette #1 into drive 0.

Insert the diskette with GRAF/BAS into drive 1.

Enter:

BASCOM
GRAF:1,GRAF:1=GRAF:1

USE: (where name is filename)

BASCOM name:1=name:1-E

when a /LST file is not needed or is too long to allow room on disk for the /CHN file which is required.

BASCOM is loaded into memory and will begin compiling the source file GRAF/BAS, converting the source program statements into object code. The object code is stored in a disk file named GRAF/REL on the disk in the drive (for multi-drive systems the file is saved on the disk in drive 1). At the same time, a listing file named GRAF/LST is saved on the disk.

6. Look for error messages.

When BASCOM has finished, program control will be returned to TRSDOS, just as in step 2 above. Since all errors should have been corrected previously, No error messages should appear on the screen, and the number preceding FATAL ERRORS should be 00000. If not, you will have to repeat steps 4, 5, and 6.

7. Enter DIR. (or DIR :1 for multi-drive systems)

You should see two new files listed in the directory: GRAF/REL and GRAF/LST.

8. List the GRAF/LST listing file.

Enter the command LIST GRAF/LST. Your Model I will display the listing file on the screen. Since the listing will be shown too fast to read, use the <SHIFT><@> (depressed at the same time) to suspend the listing while you read it. Depressing any other key will start the listing again.

As an alternative, you may have the listing file printed out on a line printer. To do this, enter PRINT GRAF/LST.

9. Review the following listing.

The following listing is a printout of the file GRAF/LST.

Notice that each line of the source program is listed, followed by the code generated for that line. The code includes CALLs to machine language subroutines in the runtime library or runtime module as well as assembly language operations.

Also notice the two hexadecimal numbers preceding each line from the source program. The first number is the relative address of the code associated with that line, using the start of the program as 0. The second number is the cumulative data area needed so far during the compilation. These two columns are totalled at the end of the listing. The left column total is the actual size of the /REL file generated (in this example, the number 01BE). The right column total is the total data area required (in this example, the number 012C).

```
BASCOM X.XX - COPYRIGHT 1979, 80 (C) BY MICROSOFT - 13527 BYTES FREE

 0014 0007        100  DEFINT A-Z:DIM C(128)
          ** 0014'          CALL     $4.0
          ** 0017'L00100:
 0017 0109        200      CLS
          ** 0017'L00200: CALL     $CLS
 001A 0109        300  PRINT @0, "RANDOM NUMBERS, UNIFORM = 1, NORMAL = 6
(NEG NO. TO STOP)"
          ** 001A'L00300: LD       HL,0000
          ** 001D'         CALL     $PAT
          ** 0020'         CALL     $PR0A
          ** 0023'         LD       HL,<CONST>
          ** 0026'         CALL     $PV2D
 0029 0109        400  INPUT "SELECTION ";N:IF N<1 THEN END
          ** 0029'L00400: LD       HL,<CONST>
          ** 002C'         CALL     $IN0A
          ** 002F'         DB       00
          ** 0030'         JP       Z,I00000
          ** 0033'         CALL     $IPUA
          ** 0036'         DB       01
          ** 0037'         DB       04
          ** 0038'         LD       HL,N%
          ** 003B'         CALL     $IPUB
          ** 003E'I00000:
          ** 003E'         LD       HL,(N%)
          ** 0041'         LD       DE,FFFF
          ** 0044'         LD       A,H
          ** 0045'         RLA
          ** 0046'         JR       C,02
          ** 0048'         ADD      HL,DE
          ** 0049'         ADD      HL,HL
          ** 004A'         JP       NC,I00001
          ** 004D'         CALL     $END
          ** 0050'I00001:
 0050 010B        500  CLS:PRINT @20, "HIT ANY KEY FOR MENU"
          ** 0050'L00500: CALL     $CLS
          ** 0053'         LD       HL,0014
          ** 0056'         CALL     $PAT
          ** 0059'         CALL     $PROA
          ** 005C'         LD       HL,<CONST>
          ** 005F'         CALL     $PV2D
 0062 010B        600  FOR I = 0 TO 127
          ** 0062'L00600: LD       HL,0000
          ** 0065'         LD       (I%),HL
          ** 0068'I00002:
 0068 010D        700  C(I) = 48
          ** 0068'L00700: LD       HL,(I%)
          ** 006B'         ADD      HL,HL
          ** 006C'         LD       DE,C%
          ** 006F'         ADD      HL,DE
          ** 0070'         LD       DE,0030
          ** 0073'         LD       (HL),E
          ** 0074'         INC      HL
          ** 0075'         LD       (HL),D
```

```
0076 010D        800  NEXT I
         ** 0076'L00800: LD      HL,(I%)
         ** 0079'        INC     HL
         ** 007A'        LD      (I%),HL
         ** 007D'        LD      HL,(I%)
         ** 0080'        LD      DE,FF80
         ** 0083'        LD      A,H
         ** 0084'        RLA
         ** 0085'        JR      C,02
         ** 0087'        ADD     HL,DE
         ** 0088'        ADD     HL,HL
         ** 0089'I00001: JP      C,I00001
008C 010D        900  A$ = INKEY$:IF A$ <> "" THEN 200
         ** 008C'L00900: CALL    $INK
         ** 008F'        LD      DE,A$
         ** 0092'        CALL    $SASA
         ** 0095'        LD      DE,A$
         ** 0098'        LD      HL,<CONST>
         ** 009B'        CALL    $NETA
         ** 009E'        DW      L00200
00A0 0110        1000 SUM = 0
         ** 00A0'L01000: LD      HL,0000
         ** 00A3'        LD      (SU%),HL
00A6 0112        1100 FOR J = 1 TO N
         ** 00A6'L01100: LD      HL,(N%)
         ** 00A9'        LD      (10F%),HL
         ** 00AC'        LD      HL,0001
         ** 00AF'        LD      (J%),HL
         ** 00B2'I00004:
00B2 116         1200 SUM = SUM + RND(128)-1
         ** 00B2'L01200: LD      HL,(SU%)
         ** 00B5'        CALL    $CISA
         ** 00B8'        CALL    $STMA
         ** 00BB'        DB      81
         ** 00BC'        LD      HL,<CONST>
         ** 00BF'        CALL    $RA4
         ** 00C2'        CALL    $FADE
         ** 00C5'        DB      81
         ** 00C6'        CALL    $FADC
         ** 00C9'        DW      <CONST>
         ** 00CB'        CALL    $CINC
         ** 00CE'        LD      (SU%),HL
00D1 0116        1300 NEXT J
         ** 00D1'L01300: LD      HL,(J%)
         ** 00D4'        INC     HL
         ** 00D5'        LD      (J%),HL
         ** 00D8'        LD      HL,(J%)
         ** 00DB'        EX      DE,HL
         ** 00DC'        LD      HL,(10F%)
         ** 00DF'        LD      A,D
         ** 00E0'        XOR     H
         ** 00E1'        LD      A,H
         ** 00E2'        JP      M,I00005
         ** 00E5'        LD      A,L
         ** 00E6'        SUB     E
```

```
        ** 00E7'          LD      A,H
        ** 00E8'          SBC     D
        ** 00E9'I00005:   RLA
        ** 00EA'          JP      NC,I00004
00ED 0116       1400 RN = SUM/N
        ** 00ED'L01400:   LD      HL,(SU%)
        ** 00F0'          CALL    $CISA
        ** 00F3'          CALL    $STMA
        ** 00F6'          DB      81
        ** 00F7'          LD      HL,(N%)
        ** 00FA'          CALL    $CISA
        ** 00FD'          CALL    $DVDE
        ** 0100'          DB      81
        ** 0101'          CALL    $CINC
        ** 0104'          LD      (RN%),HL
0107 0118       1500 C(RN) = C(RN)-1
        ** 0107'L01500:   LD      HL,(RN%)
        ** 010A'          ADD     HL,HL
        ** 010B'          PUSH    HL
        ** 010C'          LD      DE,C%
        ** 010F'          ADD     HL,DE
        ** 0110'          LD      E,(HL)
        ** 0111'          INC     HL
        ** 0112'          LD      D,(HL)
        ** 0113'          EX      DE,HL
        ** 0114'          DEC     HL
        ** 0115'          LD      (T:01),HL
        ** 0118'          POP     HL
        ** 0119'          LD      DE,C%
        ** 011C'          ADD     HL,DE
        ** 011D'          PUSH    HL
        ** 011E'          LD      HL,(T:01)
        ** 0121'          EX      DE,HL
        ** 0122'          POP     HL
        ** 0123'          LD      (HL),E
        ** 0124'          INC     HL
        ** 0125'          LD      (HL),D
0126 0118       1600 IF C(RN) < 0 THEN 200
        ** 0126'L01600:   LD      HL,(RN%)
        ** 0129'          ADD     HL,HL
        ** 012A'          LD      DE,C%
        ** 012D'          ADD     HL,DE
        ** 012E'          LD      E,(HL)
        ** 012F'          INC     HL
        ** 0130'          LD      D,(HL)
        ** 0131'          PUSH    DE
        ** 0132'          POP     HL
        ** 0133'          ADD     HL,HL
        ** 0134'          JP      C,L00200
0137 0118       1700 SET (RN,C(RN))
        ** 0137'L01700:   LD      HL,(RN%)
        ** 013A'          ADD     HL,HL
        ** 00F7'          LD      DE,C%
        ** 013E'          ADD     HL,DE
        ** 013F'          LD      E,(HL)
```

```
          ** 0140'          INC     HL
          ** 0141'          LD      D,(HL)
          ** 0142'          LD      HL,(RN%)
          ** 0145'          PUSH    DE
          ** 0146'          EX      DE,HL
          ** 01147          POP     HL
          ** 0148'          CALL    $GST
 014B 0118        1800 GOTO 900
          ** 014B'L01800: JP        L00900
 014E 0118
          ** 014E'          CALL    $END
 01BE 012C

00000 FATAL ERROR(S)
12949 BYTES FREE
```

10. Delete the listing file.

Once you have reviewed the GRAF/LST file (or have printed it out) and the compilation contains no fatal errors, you should delete the listing file. The /LST file is very large because it contains both the /BAS and the /REL files. Since the /LST file is not needed except for debugging, it is best to delete it now and gain the additional disk space.

Enter KILL GRAF/LST

Further information about /LST files is given in Chapter 3.

## 2.3 LINK LOADING

You are now ready for the next procedure--Link Loading.

Link loading is accomplished using the LINK-80 Linking Loader (the file named L80/CMD on BASCOM diskette #1).

Using LINK-80 is very much like using BASIC Compiler. So, some of the steps and descriptions for LINK-80 should sound familiar. However, you should be careful not to confuse the two procedures.

The basic steps in link loading are:

1. Insert diskette(s) in drive(s)

| MULTI-DRIVE SYSTEM | ONE-DRIVE SYSTEM |
|---|---|
| Leave your program diskette in drive 1, and your BASCOM diskette #1 in drive 0. | Remove your program diskette from the drive. Insert BASCOM diskette #1 in the drive. |

2. Load LINK-80.

Loading LINK-80 is exactly like loading BASCOM. Simply type:

ACTION # 4

```
L80
```

Your TRS-80 will search diskette #1 for LINK-80, load it, then return the asterisk (*) prompt.

If you want to stop the link loading process, and you have entered only L80 and nothing more, you can exit to TRSDOS by pressing the <BREAK> key. Otherwise, you have to reboot using the <RESET> key.

3. Insert your TRSDOS diskette.

ONE-DRIVE SYSTEM

Remove BASCOM diskette #1 from your disk drive and insert your TRSDOS diskette which contains your compiled program on it.

TWO-DRIVE SYSTEM

Remove BASCOM diskette #1 from drive 0; insert BASCOM diskette #2 in drive 0. Leave your program diskette in drive 1.

THREE-DRIVE SYSTEM

Leave BASCOM diskette #1 in drive 0 and your program diskette in drive 1, and insert BASCOM diskette #2 in drive 2.

4. Enter the filename(s) you want loaded and linked.

(The letters -N-E that are attached to GRAF in the sample commands below tell BASCOM to Execute, to Save the compiled file, and to Exit to TRSDOS.)

ACTION # 5

| ONE-DRIVE SYSTEM | MULTI-DRIVE SYSTEM |
|---|---|
| Enter GRAF<br><br>When LINK-80 returns the asterisk (*) prompt, remove your TRSDOS diskette from the drive, insert BASCOM diskette #2, and enter:<br><br>BASLIB-S<br><br>When the asterisk (*) prompt appears again, remove diskette #2 from the drive and reinsert your TRSDOS program diskette into the drive. Enter:<br><br>GRAF-N-E | Leave your diskettes arranged in the disk drives as instructed in step 3 above.<br><br>Enter GRAF,GRAF:1-N-E<br><br>(Combine action 4 & 5)<br><br>Enter L80 and at * prompt<br>Enter name,name:1-N-E<br><br>where name is filename |

5. Wait.

The link loading process requires several minutes. During this time, the following messages will appear on your screen:

```
DATA     8A00     8DED     < 1005>

29784 BYTES FREE
(8B2C     8DED)

DOS READY
_
```

The value of the numbers displayed will depend on the size of your program and the amount of data. The 8DED represents the address at the top of the stored program. The 1005 represents the number of bytes of memory needed to store the program. The 29784 represents the number of bytes of memory still free.

6. Enter DIR. (or DIR :1 for multi-drive systems)

You should see a file named GRAF/CHN. This is an executable file.

## 2.4 RUNNING A COMPILED PROGRAM

Once you have compiled your program, it is a very simple procedure to run it.

1. From TRSDOS command level, follow the steps for your configuration.

ACTION # 6 FINAL RUN

| ONE-DRIVE SYSTEM | MULTI-DRIVE SYSTEM |
|---|---|
| Insert BASCOM diskette #2 (BASLIB/BRUN) into the drive.<br><br>Enter BRUN<br><br>The asterisk (*) prompt will appear. Remove diskette #2 from the drive and reinsert your TRSDOS program diskette into the drive.<br><br>Enter GRAF | Insert your executable program diskette in one drive and BASCOM diskette #2 (BASLIB/BRUN) in another drive.<br><br>Enter BRUN GRAF<br><br>Enter BRUN name<br>where name is filename |

The program should now run just as it does under the Disk BASIC interpreter, but the program should run much faster. Try returning to Disk BASIC and running GRAF/BAS under the interpreter. Then, from TRSDOS, run the compiled version of GRAF. If you input the same number for each run, you should see that the compiled version runs about three times faster than the interpreter version.

## 2.4.1 Sample BRUN Output Of GRAF/CHN

This sample program is a user-interactive program. The initial output that will appear in the upper left-hand corner of your screen is:

```
     RANDOM NUMBERS, UNIFORM = 1, NORMAL = 6 (NEG NO. TO
STOP)
SELECTION ?
```

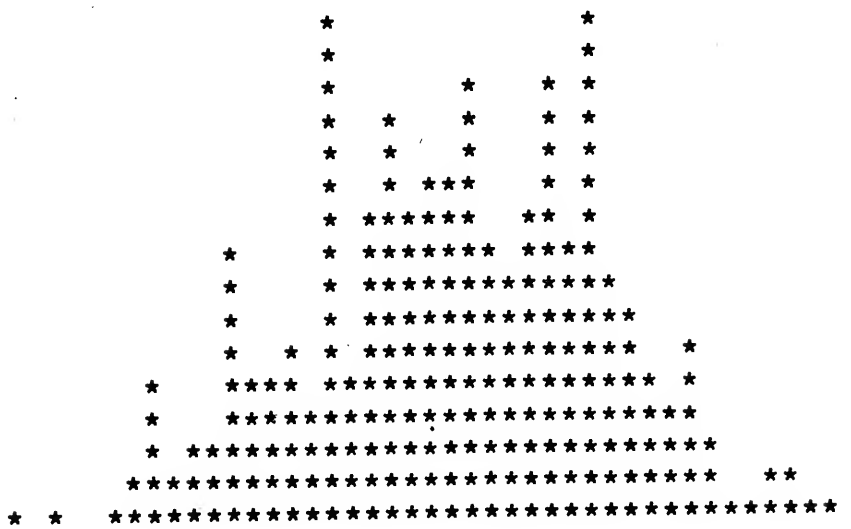
When you see this, enter any integer between 1 and 128 and the graph should appear as shown for the two samples.

```
                    HIT ANY KEY FOR MENU

   *      *
   *      *
 * *      *
 * *      *
 * *      *     *
 * **     *     *      *        *                    *      *          *
 ****     *     *      *      * *        *           *      *     *    *
 ****     *     *     **   *   * *     *  **       * **   * ***    * *   *
 **** * **    ** *   **   ** * *     *   **    * ***** ******* * ** **
 **** * ***   ****   *** ** * **     *   ** * ******* ********* ** **
 **** * **** **** **** ** ****   **   **** ****************** *****
 **** *********** ******* **** ***********************************
 **** *********** *********** ***********************************
 ****************************************************************
 ****************************************************************
 ****************************************************************
```

RANDOM NUMBERS, UNIFORM = 1, NORMAL = 6 (NEG NO. TO STOP)
SELECTION ? 6

HIT ANY KEY FOR MENU



## LEARN MORE ABOUT USING BASIC COMPILER

You have successfully compiled and run a simple BASIC program. Now, you are ready to learn the technical details you need to know to compile your own more complex programs. Chapter 3 contains more extensive descriptions of each of the steps you followed in this chapter, including all of the options you can specify to the BASIC Compiler program commands. Chapter 3 also describes all of the language, operational, and feature differences between BASIC Compiler and your BASIC interpreter. (These differences are summarized in Table 3.0 at the beginning of Chapter 3.) Be sure you have read this material before you attempt to compile your own programs.

# CHAPTER 3

# TECHNICAL DESCRIPTIONS

Chapter 2 presented a brief explanation of each step in the compilation of a program. This chapter explains each step in greater detail, including all the different options available to you in the compiling process.

## 3.1 EDITING

IMPORTANT: BASIC programs which you want to compile are, for the most part, written the same way you write programs to run with the interpreter (Disk BASIC). However, there are certain language differences between BASIC Compiler and BASIC interpreter that must be taken into account when compiling existing or new BASIC programs. That is why we strongly recommend that you compile the sample program in Chapter 2 first, read Chapter 3 second, and only then begin compiling your own programs.

These differences fall into three categories: operational differences, language differences, and feature differences.

The table on the next page is a reference guide to the feature differences between the interpreters and BASIC Compiler. The differences are explained in detail in the text following the table.

The symbols used in the table indicate:

Y = feature is supported by BASIC Compiler and executes the same as in the interpreter

N = feature is not supported by BASIC Compiler

D = feature is supported by BASIC Compiler but there are some differences in execution between BASIC Compiler and the interpreter. These differences are described in this chapter.

X = manual where the feature is described

| Feature | BASIC Compiler | Disk BASIC Manual | Level II BASIC Reference Manual | BASIC-80 Reference Manual |
|---|---|---|---|---|
| AUTO | N | | X | X |
| CALL | Y | | | X |
| CHAIN | D | | | X |
| CLEAR | N | | X | X |
| CLS | Y | | X | |
| CLOAD | N | | X | X |
| CMD"R" | Y | X | | |
| CMD"S" | Y | X | | |
| CMD"T" | Y | X | | |
| COMMON | N | | | X |
| CONT | N | | X | X |
| CSAVE | N | | X | X |
| DEFxxx | D | | X | X |
| DELETE | N | | X | X |
| DIM | D | | X | X |
| EDIT | N | | X | X |
| END | D | | X | X |
| ERASE | D | | | X |
| FOR/NEXT | D | | X | X |
| IN#-1 | N | | X | |
| INKEY$ | Y | | X | |
| %INCLUDE | Y,X | | | |
| LIST | N | X | X | X |
| LLIST | N | | | X |
| LOAD | N | X | | X |

| Feature | BASIC Compiler | Disk BASIC Manual | Level II BASIC Reference Manual | BASIC-80 Reference Manual |
|---|---|---|---|---|
| MEM | Y | | X | |
| MERGE | N | X | | X |
| NEW | N | | X | X |
| | | | | |
| ON ERROR GOTO | D | | X | X |
| OUT#-1 | N | | X | |
| POINT(x,y) | Y | | X | |
| PRINT @ | Y | | X | |
| RANDOM | Y | | X | |
| RANDOMIZE | N | | | X |
| RENUM | N | | | X |
| RESET(x,y) | Y | | X | |
| RUN | D | X | X | X |
| SAVE | N | X | | X |
| SET | Y | | X | |
| STOP | D | | X | X |
| SYSTEM | N | | X | |
| TIME$ | Y | X | | |
| TRON/TROFF | D | | X | X |
| USRn | D | | X | X |
| WHILE/WEND | D | | | X |

Table 3.0: Feature Differences between Interpreter and Compiler

### 3.1.1 OPERATIONAL DIFFERENCES

Normally, the Compiler interacts with the console only to read compiler commands. These commands specify only what files are to be compiled and some limited manipulations of the compiling process. So, commands that are usually issued in direct mode with Disk BASIC are not implemented for BASIC Compiler. The following statements and commands are not implemented and, if used, all but SYSTEM generate error messages.

| | | | | | |
|---|---|---|---|---|---|
| AUTO | CLEAR | CLOAD | CSAVE | CONT | DELETE |
| EDIT | LIST | LLIST | RENUM | SAVE | LOAD |
| MERGE | NEW | COMMON | IN#-1 | OUT#-1 | SYSTEM |

NOTE: BASIC Compiler does not currently support any cassette I/O.

While the interpreter cannot accept a physical line that is more than 254 characters in length, its logical statements may contain as many physical lines as desired. In contrast, BASIC Compiler accepts logical lines up to 253 characters in length.

### 3.1.2 LANGUAGE DIFFERENCES

Most programs that run under the TRS-80 Disk BASIC interpreter will compile under the BASIC Compiler with little or no change. However, it is necessary to note differences in the use of the following program statements:

1. CALL
   The CALL statement allows you to call and transfer flow to a MACRO-80 or TRS-80 FORTRAN subroutine. The format of the CALL Statement is:

   CALL <variable name> [<argument list>...]

   where <variable name> and <argument list> are supplied by the programmer.

<variable name> is the name of the subroutine the programmer wishes to call. This name must be 1 to 6 characters long and must be recognized by LINK-80 as a global symbol. That is, <variable name> must be the name of the subroutine in a FORTRAN SUBROUTINE statement or a PUBLIC symbol in an assembly language routine. (Refer to the MACRO-80 manual or FORTRAN-80 manual for definitions of these terms. See NOTE below.)

<argument list> is optional and contains the arguments that are passed to the assembly language or FORTRAN subroutine.

Example: 120 CALL MYSUBR (I,J,K)

NOTE

Assembly Language Development System (of which MACRO-80 is a part) and FORTRAN-80 are separate products from BASIC Compiler and are not necessarily associated with BASIC Compiler. If you do not have one of these products, the CALL statement cannot be used. Both products are available from Microsoft Consumer Products.

2. COMMON
The COMMON statement is not implemented in the compiler. It will generate a fatal error.

3. CHAIN and RUN
The CHAIN and RUN statements have been implemented in their simplest form only; i.e., CHAIN <string expression> and RUN <string expression>. For TRS-80, the default extension is /CHN. BASCOM programs can chain to any CHN file.

4. DEFINT/SNG/DBL/STR
DEFxxx statements are directions to the compiler, telling the compiler how to code the variables listed after the DEFxxx statement. So, the compiler does not "execute" DEFxxx statements as it does a FOR statement, for example. Rather, the compiler reacts to the occurrence of these statements, regardless of the order in which program lines are executed, by setting the way the defined variables will be coded when encountered--as INTegers, as SiNGle precision floating point, as DouBLe precision floating point, or as STRing variables.

A DEFxxx statement takes effect as soon as its line is encountered. Once the type has been defined for the listed variables, that type remains in effect either until the end of the program or until a different DEFxxx statement which lists the same variables is encountered. For variables given with a precision designator (i.e., %, !, #, as in A%=B), the type will not be affected by a DEFxxx statement.

5. DIM and ERASE

The DIM statement is similar to the DEFxxx statement in that it is scanned rather than executed. That is, DIM takes effect when its line is encountered and remains in effect until the end of the program. If the default dimension (10) has already been established for an array variable and that variable is later encountered in a DIM statement, an "Array Already Dimensioned" error results. Therefore the common practice of putting a collection of DIM statements in a subroutine at the end of the program will generate fatal errors. The compiler will see the DIM statement only after it has already assigned the default dimension to variables encountered earlier in the program.

There is no ERASE statement in the compiler, so arrays cannot be erased and redimensioned. If the compiler encounters an ERASE statement, a fatal error is produced.

Also note that the values of the subscripts in a DIM statement must be integer constants; they may not be variables, arithmetic expressions, or floating point values. For example,

```
DIM A1(I)
DIM A1(3+4)
DIM A1(3.4E5)
```

are all illegal.

6. END

During execution of a compiled program, an END statement closes files and returns control to the operating system. The compiler assumes an END statement at the end of the program, so "running off the end" (omitting an END statement at the end of the program) produces proper program termination by default.

7. FOR/NEXT

FOR/NEXT loops must be statically nested. Static nesting means that each FOR must have a single corresponding NEXT. Static nesting also means that each FOR/NEXT pair must be inside the previously encountered pair. So, for example:

```
FOR I        FOR I          FOR I
FOR J        FOR J          FOR J
FOR K        FOR K          FOR K
 :     and     :              :
 :             :              :
NEXT J       NEXT           NEXT K
NEXT K                      NEXT J
NEXT I                      NEXT I
```

are not allowed.

is the correct form.

Also not allowed is a statement that directs program flow into a FOR/NEXT loop without entering through the FOR statement. For example:

```
50  GOTO 100
        :
        :
90  FOR I
        :
        :
200 NEXT I
```

is not allowed.

8. %INCLUDE

The %INCLUDE <filename> statement allows the compiler to include source code from an alternate BASIC file. These BASIC source files may be subroutines, single lines, or any type of partial program. No assembly language or FORTRAN files are allowed as argument to the %INCLUDE statement.

The programmer should take care that any DIM statements or variables in the included files match their counterparts in the main program, or that included lines do not contain GOTOs to non-existant lines, END statements, or similarly erroneous code.

Three restrictions must be observed:

(1) The included lines must be in ascending order.
(2) The lowest line number of the included lines must be higher than the line number of the INCLUDE statement in the main program.

(3) The last line number of the included lines must be lower than the line number in the main program which will follow the included lines. (These restrictions would be removed if the main program were compiled with the -C switch set--see Command Line Switches under Section 3.2 BASIC COMPILER in Chapter 3).

The %INCLUDE statement must be the last statement on a line. The format of the %INCLUDE statement is:

<line number> %INCLUDE <filename>

For example,

999 %INCLUDE SUB1000/BAS

9. ON ERROR GOTO/RESUME <line number>
If a program contains ON ERROR GOTO and RESUME <line number> statements, the -E compilation switch must be given in the compiler command line. If the RESUME NEXT, RESUME, or RESUME 0 form is used, the -X switch must be used instead.

The basic function of these switches is to allow the compiler to function correctly when error trapping routines are included in a program. See BASIC Compiler Switches, Section 3.2.3 below for a detailed explanation of these switches. Note, however, that the use of these switches greatly increases the size of the /REL and /CHN files.

10. REM
REM statements are REMarks starting with a single quotation mark or the word REM. Since REM statements do not take up time or space during execution, REM may be used as freely as desired.

11. STOP
The STOP statement is identical to the END statement. Open files are closed and control returns to the operating system.

12. TRON/TROFF
In order to use TRON/TROFF, the -D compilation switch must be issued in the compiler command line. Otherwise, TRON and TROFF are ignored and a warning message is generated.

13. USRn Functions

Although the USRn function is implemented in the compiler to call machine language subroutines, there is no way to pass parameters, except through the use of POKEs to protected memory locations that are later accessed by the machine language routine.

When the compiler sees X = USRn (0), it generates code for

```
CALL    $U%+CONST
LD      (X%),HL
```

During execution, the program encounters this code, jumps to the address of the CALL, performs the steps of your subroutine and returns. Your routine should place the integer result of the routine in H,L prior to returning to the compiled BASIC program. On return, as shown above, the contents of the H,L reister pair is placed in the variable X. Any other parameters to be passed back must be stored in protected memory locations and PEEKed from the main BASIC program. Using this method of passing parameters, the USRn function is quite usable. The programmer must take responsibility, though, to protect his machine code routine. This is more complicated than in the interpreter because the top of memory pointer at 4049H and 404AH cannot be set from within the compiled program. This pointer, therefore, must be set prior to executing the compiled program if any part of high memory is to be protected.

Programmers not wishing to do this have two choices. If your machine language routine is short enough, you can store it by making the first string defined in the program be the ASCII values corresponding to the hex values of the routine. Use the CHR$ function for this. You can then find the start of your routine by using the VARPTR function. For example, for the string A$, VARPTR (A$) will return the address of the length of the string. The next two addresses will be (first) the least significant byte and (then) the most significant byte of the actual address of the string.

The second method is to reset the default value of the linking loader (normally 8A00H) with the -P switch in the command line. By setting the address for the start of the loading area to 8B00H, for example, 256 bytes of free protected space are created between the end of BRUN and the start of the loading area. Machine language routines or

data can then be safely POKEd into this area.

A better alternative is to use MACRO-80 (part of the Assembly Language Development System available from Microsoft Consumer Products) to assemble your subroutines. Then, your subroutines can be linked directly to the compiled program and referenced using the CALL statement as defined above in item 1.

14. WHILE/WEND

WHILE/WEND loops must be statically nested. Static nesting means that each WHILE must have a single corresponding WEND. Static nesting also means that each WHILE/WEND pair must be inside the previously encountered pair. Refer to the example shown for the FOR/NEXT statements above.

Also not allowed is a statement that directs program flow into a WHILE/WEND loop without entering through the WHILE statement. See FOR/NEXT above for an example of this restriction also.

Also, FOR/NEXT and WHILE/WEND loops may not cross boundaries with each other.

```
FOR

WHILE

NEXT

WEND
```

is not allowed, or any similar arrangement.

### 3.1.3 FEATURE DIFFERENCES

The differences in features between BASIC-80, Level II BASIC, Disk BASIC, and BASIC Compiler fall into several categories. (See Table 3.0 at the beginning of Chapter 3 for a summary of the differences.) First, BASIC Compiler supports some statements and functions found in Disk BASIC but which are not described in the BASIC-80 Reference Manual. Second, BASIC Compiler supports some additional functions which are not supported by Disk BASIC (and therefore do not function if you try to run them under the interpreter). Third, BASIC Compiler's handling of expression evaluation differs from the interpreter's capabilities. And fourth, the compiler's ability to make use of integer variables allows it to execute certain

functions up to 30 times faster than the interpreter.

## DISK BASIC COMMANDS NOT FOUND IN THE BASIC-80 REFERENCE MANUAL

Microsoft BASIC Compiler supports some statements and functions of Disk BASIC that are not described in the BASIC-80 Reference Manual. These are documented in the Level II BASIC and in the TRSDOS and Disk BASIC manuals. The following statements and functions execute in BASIC Compiler just as they do in Disk BASIC.

Statements

```
CLS
CMD"R"
CMD"R" [,"dd/mm/yy hh:mm:ss"]
CMD"S"
CMD"T"
PRINT @pos,...
RANDOM
RESET(x,y)
SET(x,y)
```

Functions

```
INKEY$
MEM
POINT(x,y)
TIME$
```

## BASIC COMPILER FEATURES NOT IN DISK BASIC

The BASIC Compiler supports many features not found in Disk BASIC that can add power and efficiency to your programming. Keep in mind that these new features compile with no problems, but you cannot run a program using these features with your interpreter, because Disk BASIC does not recognize them.

1. Double Precision Transcendental Functions
   SIN, COS, TAN, SQR, LOG, and EXP return double precision results if given a double precision argument. Exponentiation with double precision operands will return a double precision result.

2. Long Variable Names
   Variable names may be up to 40 characters long with all 40 characters significant. Letters, numbers, and the decimal point are allowed in variable names, but the name must begin with a letter.

Variable names may also include embedded reserved words. Reserved words include all BASIC-80 commands, statements, function names, and operator names.

NOTE

To take advantage of the long variable name capability, you must specify the -5 switch when you enter the compiler command line. Refer to Section 3.2.3, BASIC Compiler Switches, for details.

3. Fixed Stack
BASIC Compiler uses a 256 byte fixed stack at the top of memory. Consequently, the programmer cannot branch indefinitely. For every GOSUB issued, the program must execute a RETURN. Nesting is allowed, but only up to 100 levels. If this limit is not observed, your program will crash.

## EXPRESSION EVALUATION

During expression evaluation, BASIC Compiler converts the operands of each operator to the same type, that of the more precise operand. For example,

```
QR=J%+A!+Q#
```

causes J% to be converted to single precision and added to A!. This result is converted to double precision and added to Q#.

BASIC Compiler is more limited than the interpreter in handling numeric overflow. For example, when run on the interpreter the program

```
I%=20000
J%=20000
K%=-30000
M%=I%+J%-K%
```

yields 10000 for M%. That is, it adds I% to J% and, because the number is too large, it converts the result into a floating point number. K% is then coverted to floating point and subtracted. The result of 10000 is found and is converted back to integer and saved as M%.

BASIC Compiler, however, must make type conversion decisions

during compilation. It cannot defer until the actual values are known. Thus, the compiler would generate code to perform the entire operation in integer mode. If the -D switch were set, the error would be detected. Otherwise, an incorrect answer would be produced.

In order to produce optimum efficiency in the compiled program, the compiler may perform any number of valid algebraic transformations before generating the code. For example, the program

```
I%=20000
J%=-18000
K%=20000
M%=I%+J%+K%
```

could produce an incorrect result when run. If the compiler actually performs the arithmetic in the order shown, no overflow occurs. However, if the compiler performs I%+K% first and then adds J%, an overflow will occur. The compiler follows the rules for operator precedence and parentheses may be used to direct the order of evaluation. But, no other guarantee of evaluation order can be made.

## INTEGER VARIABLES

To produce the fastest and most compact object code possible, make maximum use of integer variables. For example, this program

```
FOR I=1 TO 10
A(I)=0
NEXT I
```

can execute approximately 30 times faster by simply substituting "I%" for "I". It is especially advantageous to use integer variables to compute array subscripts. The generated code is significantly faster and more compact.

## 3.2 BASIC COMPILER

What is a BASIC Compiler? Perhaps the best way to understand the BASIC Compiler is to examine how program execution takes place. The microprocessor in your computer can execute only a very narrow range of instructions known as machine language or machine code. BASIC, and other programming languages, were designed to make programming easier, more readable. Since the microprocessor cannot execute BASIC program statements, some type of translation must occur. Compilers and interpreters are two types of system programs which perform this translation.

An interpreter, like TRS-80 Disk BASIC, performs the translation line by line. Every time the interpreter executes a BASIC statement it must break down the statement, check it for errors, and translate the statement into a series of machine language instructions and appropriate CALLs to subroutines that perform the BASIC function requested. Interpreters are fairly fast, but these processes all take time. If the line must be executed again, the entire process must be repeated. Also, since BASIC lines may not be in the same absolute place in memory each time, (if you insert a new line for example) branches such as GOTOs and GOSUBs cause the interpreter to examine every line number, starting at the beginning of the program, until it finds the line you referenced.

A compiler, on the other hand, takes a source program and translates it into a machine language program, called the object program or object file, that the microprocessor can execute directly. Since the translation only takes place once, and all of it prior to execution, the execution is speeded tremendously. In most cases this will be about 3 to 10 times faster than the TRS-80 Disk BASIC interpreter. And if maximum use of integer variables is made, it can be up to 30 times faster.

Most compilers have a general purpose text editor which the programmer uses to type in the source program. The compiler is then applied to this source program in a process known as the compile step or simply compile time. Syntax errors in the source file generally show up during compile time. An error free source file can then be loaded and linked if necessary and then executed. The execution phase is called runtime. Programming logic errors will show up during runtime.

With most compilers, compile time errors and runtime errors must be corrected by restarting the editor and altering the source file. Testing the changes requires repeating the entire compilation process. With Microsoft's BASIC Compiler, the process is similar, with one advantage: the TRS-80 Disk BASIC interpreter becomes your editor. If your program does not use the special features of BASIC Compiler

and care is taken to follow BASIC Compiler syntax in the few areas of difference, the program can be tested and debugged before it is compiled, saving program development time.

The Microsoft BASIC Compiler produces its output in relocatable binary format, which must then be loaded into its final area of execution using the LINK-80 linking loader. LINK-80 is described in Section 3.3.

### 3.2.1 BASIC Compiler Command Line

A BASCOM command line has three fields or parts. Each field (when filled with an entry) commands BASCOM to perform a specific function. (When no entry is made in a field, the function is omitted.) Later we will discover that each field may have several parts, too. For now, let's get an overview of each field and its usual entry.

The three fields are named for the type of content entered into them. As seen in command line format, the three fields look like this:

```
[objectfile][,listingfile]=sourcefile
```

Let's discuss the fields in order of importance.

SOURCEFILE FIELD (=sourcefilename)

The sourcefile field is the most important. Even though the other two fields may be omitted, the sourcefile field cannot. Also, the equal sign (=) must always be included as a part of the sourcefile field.

Therefore, the minimum command line to BASCOM is: =sourcefile name. The sourcefile field entry is always the name of the BASIC file you want compiled. Using our sample program from Chapter 2, the minimum command line is

```
=GRAF
```

If you enter only =GRAF, BASCOM will compile the GRAF source program, but no object code file and no listing file will be created. Simply typing =sourcefile allows you to check your program for syntax errors before you save the compiled program. The compiler operates faster in this mode since less disk I/O is performed, allowing for fast error checking.

OBJECTFILE FIELD (objectfilename)

(NOTE: In a command line which contains a comma, the comma is part of the listingfile field, not part of the objectfile field.)

The objectfile field may have an entry or may be omitted.

When the objectfile field has no entry, no object code file is created. Compilation occurs, but the compiled code is not saved on disk.

When the objectfile field has an entry, an object code file is created and saved on disk under the name entered in the objectfile field. The filename is automatically given the extension /REL unless otherwise specified. It is best, however, to let the compiler give the filename the /REL extension. (Refer to the discussion of Filename Extensions below.)

The command line

```
GRAF=GRAF
```

commands BASCOM to compile the source file GRAF/BAS, then create a file named GRAF/REL to save the object code. When control has exited from BASCOM to TRSDOS, enter DIR, and you should see a file named GRAF/REL.

LISTINGFILE FIELD (,listfilename)

The listingfile field, like the objectfile field, may have an entry or may be omitted.

Whenever the listingfile field is omitted, no listing file is created. Therefore, you have no way of reviewing the code generated by the compiler.

When a listingfile field entry is given, it can be a filename or a device name. If it is a filename, a file is created and saved on disk under that name with the extension /LST unless otherwise specified. If the listfield entry is a device, such as the printer (,*PR), the listing is sent to the device. After the compilation is completed, you can review the code generated by BASIC Compiler.

Entries in the listingfile field must be preceded by a comma. Again, using our GRAF program, you can create a listing file (without creating an object code file) with the command:

```
,GRAF=GRAF
  or
,*PR=GRAF
```

When control has exited from BASCOM to TRSDOS, enter DIR, and you should see a file named GRAF/LST.

To compile the GRAF/BAS program and create both an object code file and a listing file, enter

```
GRAF,GRAF=GRAF
  or
GRAF,*PR=GRAF
```

If you then enter LIST GRAF/LST, the code generated by BASCOM will be displayed on your screen.

NOTE

The list file will be displayed much too fast to read. However, if you have a printer, you can print out the list file and study the code at leisure. If you do not have a printer or would rather list the file on the screen, use <SHIFT><@> (depressed at the same time) to suspend the listing while you read it. Depressing any other key will start the listing again.

The listing file can provide you some important information.

Each line of the source program is listed followed by the code which was generated for each source line. Two hexadecimal numbers precede each BASIC line. The first number is the relative address of the code associated with that line, using the start of the program as 0. The second number is the cumulative data area needed for variables and so forth to that point in the compilation. These two columns are totalled and printed just prior to the number of fatal errors on the listing. The left column total is the actual size of the /REL file generated. The right column is the total data area required.

At the end of the listing is printed XXXXX BYTES FREE, where XXXXX is some decimal number. This figure has no direct relationship to the size of your program. The compiler constantly reads from the source disk file and writes to the object disk file. Consequently, your complete program never resides in memory. The compiler does, however, use up memory during the operation by storing long strings and certain code to avoid duplication, storing tables of data and variables, and so on. This BYTES FREE message does, therefore, give some indication of how close your program comes to exceeding the size limits of the compiler. If you exceed the limit, the compiler will stop and print 00000 BYTES FREE.
If this happens, you may still be able to compile the program by:

setting the -S switch
removing ON ERROR GOTOs
eliminating the -D switch

The relative addresses are also useful for locating the lines that generate runtime errors. Refer to Section 3.4.3, Runtime Error Messages, for details.

### 3.2.2 Other Possible Field Entries

The other possible entries you may make in a command line field fall into three categories. (There is actually a fourth category, called switches, but we shall discuss this category under a separate heading below.) The categories are filename extensions, device specifications, and interactive entries.

#### FILENAME EXTENSIONS

The programmer may select any name for files, as long as the name does not exceed eight characters and the first character is a letter A-Z. BASCOM also allows you to append an extension to the filename. Programmers add extensions to filenames as identification tags. Extensions are usually mnemonic to assist the programmer's memory in recalling the type of file.

Filename extensions may be up to three characters. Filename extensions must be separated from the filename by a slash mark (/), but no other spaces.

BASCOM supplies filename extensions automatically whenever certain types of files are created. We have already hinted at this when discussing the command line fields above. The filename extensions that are supplied automatically by BASCOM are called default filename extensions. The default extensions are:

| | |
|---|---|
| /REL | Relocatable object file - given to the file created when the objectfile field contains a name. |
| /LST | Listing file - given to the file created when the listingfile field contains a name. |
| /CHN | Executable command file - given to the file created after link loading. |

BASCOM also recognizes, by default, the filename extension

| | |
|---|---|
| /BAS | BASIC source file - given by you when you saved your BASIC program after the Editing procedures. (You must assign an extension to your filename when you save the program onto disk.) |

As you program, you may wish to give your files different extensions to distinguish them in some way from files with the default extensions. You may certainly give your files

any extension you choose. However, be aware of one or two problems. First, the default extensions were chosen for their mnemonic qualities--they aid memory. You will always immediately recognize a filename with the extension LST as a listing file. If you choose your own extensions, you may want to consider making the extensions as mnemonic as possible. Second, the BASIC Compiler programs (BASCOM, L80, and BRUN) assume that the files will have these default extensions. If you give your files different extensions, you must always remember to include the extension when you give the filename. If you forget to include the extension, BASCOM will be unable to find the file.

For example, you may enter

```
BASCOM GRAF/OBJ,GRAF/DOC=GRAF
```

BASCOM will create an object file named GRAF/OBJ and a listing file named GRAF/DOC from a source file named GRAF/BAS.

Note, however, that when you enter a LINK-80 command line, you would be required to include the /OBJ filename extension. (The filename extension /REL is omitted in LINK-80 command lines.) For example:

```
L80 GRAF/OBJ,GRAF-N-E
```

Any of the field entries may be given any filename extension you choose, as long as you always remember to include the extension as part of the filename in a BASCOM or L80 command line.

## DRIVE SPECIFICATION

If you have a multi-drive system, you can specify which drive BASCOM will search for the source file, and to which drive or device name device BASCOM will send the object and listing files. (If you have a single-drive system, the drive specification is always drive 0. So this information has no value for you.)

Your TRS-80 Model I supports up to four disk drives. These drives are always numbered :0, :1, :2, :3. The colon is part of the drive specification and must be included when specifying a drive. Of course, if you have, for example, only two drives, drives numbers :2 and :3 are not available.

The following command line gives an illustration of the use of drive specification.

```
GRAF:2=GRAF:1
```

This command line causes BASCOM to search the diskette in drive :1 for the file named GRAF/BAS, to compile the file, and to place the object code in a file named GRAF/REL on the diskette in drive :2.

For source files, the drives are searched in ascending order until the specified file is located. It is never necessary to specify the drives unless you only want to search one specific diskette, or, when saving the BASIC source program, you want to specify the diskette where the file is to be saved. For object and listing files, on the other hand, if a drive number is omitted, the files are placed on the diskette that is in drive 0.

> There is an exception to this. If the diskette in drive 0 is write protected, the drives are searched until a diskette that is not write protected is found. Also, if you are recompiling a program (that is, copying over a previously compiled /REL file), the object file will always be placed on the diskette which already contains that /REL file, regardless of the drive number that the diskette is in. This can cause a problem if you want two /REL files with the same filename and you simply put a fresh disk in drive 0 while leaving the old /REL file diskette in drive 1. TRSDOS will still try to write the new /REL file to the drive with the old /REL file, even if the diskette is write protected.

IMPORTANT: The diskette in drive 0 must always contain TRSDOS.

## INTERACTIVE ENTRIES

BASIC Compiler is not an interactive program. However, it is possible to cause BASCOM to compile lines input directly from the keyboard, to display lines on the screen as they are compiled, or to print out lines on a printer as they are compiled.

To cause BASIC Compiler to perform these functions, you must substitute device names in place of the filename you would normally use for a command line field. The device names and their uses are:

(NOTE: The asterisk (*) is a necessary part of the following entries.)

*KI (Keyboard Input) may be entered in place of source filename.

Example:

GRAF,GRAF=*KI compiles each line as it is entered and places object code in GRAF/REL and listing in GRAF/LST. The program line number must be included with each line entered.

Also, in this mode BASIC Compiler will only accept 80 characters of input without a carriage return. If you press <ENTER>, BASIC Compiler accepts the entry as the total line.

,*DO (Display Output) may be entered in place of list filename.

Example:

GRAF,*DO=GRAF displays the list file (source and compiled code) for each line on the screen as it is compiled.

,*PR (Printer Output) may be entered in place of list filename.

Example:

GRAF,*PR=GRAF prints the list file (source and compiled code) for each line on the line printer as it is compiled.

Two combinations of these special device names provide immediate output of compiled input as soon as the input is entered.

Either of these two modes can be very useful in debugging programs. If you receive several error messages during the compilation of a program and you are not sure of the correct form to use, set BASIC Compiler in this mode and type in the lines in different forms until you are sure they compile without errors.

Normally, BASCOM's exit to TRSDOS is automatic when compiling is finished. In interactive mode, however, you must press the <BREAK> key to exit to TRSDOS.

,*DO=*KI (Display Output from Keyboard Input) is entered in place of list filename and source filename. As soon as a program line is entered, BASCOM compiles the line, displays immediately the code listing on the screen, and indicates any errors.

To save the generated code, enter a filename in the objectfile field. For example,

```
GRAF,*DO=*KI
```

Note, however, that you have no means to edit this file since you are creating an object file directly from keyboard input. If you make a false entry, you must simply <BREAK> from the program, KILL the /REL file, and start over. Therefore, this method is recommended only for debugging.

,*PR=*KI (Printer Output from Keyboard Input) is entered in place of list filename and source filename. This command line causes BASCOM to function the same as *DO=*KI, except that the code is listed on the line printer instead of the screen. The same limitations and cautions apply to *PR=*KI as to *DO=*KI.

### 3.2.3 Command Line Switches

Besides specifying which files shall be created and how they shall be named, you can direct the compiler to perform some other functions as it compiles. Because you may choose to have the compiler perform these functions only sometimes, you need to be able to "switch" these functions on as needed. The means for specifying these additional functions are called Switches.

The uses for the different switches are described below. Generally, switches direct the compiler to perform additional functions. Switches are placed at the end of a compiler command line. Switches are always preceded by a dash, and more than one switch may be used in the same command. An example of the format would be:

```
GRAF,GRAF=GRAF-Z-4-T
```

(These switches are the default switch settings BASCOM uses if you don't specify any switches.)

Let's go over the compiler command line switches. First, you'll find a summary list which gives a brief description of the function(s) each switch controls. Following the summary, you'll find detailed explanations of each switch.

| SWITCH | ACTION |
|---|---|
| -E | Program has ON ERROR GOTO with RESUME<line number> |
| -X | Program has ON ERROR GOTO with RESUME, RESUME 0, or RESUME NEXT |
| -N | No object code in the listing file |
| -D | Generate debug/code checking at runtime. |
| -Z | Use Z80 opcodes (default) |
| -S | Write quoted strings to binary file |
| -4 | Use Microsoft 4.51 lexical conventions (default) (Not used with -C) |
| -5 | Use Microsoft BASIC 5.2 conventions: -5-4 together for Disk BASIC lexical but BASIC 5.2 execution conventions. -5-T together for Disk BASIC execution but BASIC 5.2 lexical conventions. |
| -C | Relax line numbering contraints. Must also specify -5. (Not used with -4) |
| -T | Use Disk BASIC execution conventions (default) |

Each of these switches is explained in detail in the following list.

| SWITCH | ACTION |
|---|---|
| -E | The -E switch tells the compiler that the program contains the ON ERROR GOTO statement. If a RESUME statement other than RESUME <line number> is used with the ON ERROR GOTO statement, use -X instead (see below). To handle ON ERROR GOTO properly in a compiled environment, BASIC must generate some extra code for the GOSUB and RETURN statements. Therefore, do not use this switch unless your program contains the ON ERROR GOTO statement. The -E switch also causes line numbers to be included in the binary file, so runtime error messages will include the number of the line in error. |
| -X | The -X switch tells the BASIC Compiler that the program contains one or more RESUME, RESUME NEXT, or RESUME 0 statements. The -E switch is assumed when the -X switch is specified (you need to set only -X even though you are using both types of RESUME statements). To handle RESUME statements properly in a compiled environment, the compiler must relinquish certain optimizations. Therefore, do not use this switch unless your program contains RESUME statements other than RESUME <line number>. The -X switch also causes line numbers to be included in the binary file, so runtime error messages will include the number of the line in error. |
| -N | The -N switch prevents listing of the generated code in symbolic notation. If this switch is not set, the listing file produced by the compiler will contain the object code generated by each statement. When listing files to a printer (,*PR), it is useful to use the -N switch. Only your program lines and the error indicators are printed, which saves considerable paper and makes the errors much easier to find. |

-D The -D switch causes debug/checking code to be generated at runtime. This switch must be set if you want to use TRON/TROFF. The BASIC Compiler generates somewhat larger and slower code in order to perform the following checks:

1. Arithmetic overflow. All arithmetic operations, integer and floating point, are checked for overflow and underflow.

2. Array bounds. All array references are checked to see if the subscripts are within the bounds specified in the DIM statement.

3. Line numbers are included in the generated binary so that runtime errors can indicate the statement which contains the error.

4. RETURN is checked for a prior GOSUB.

-Z The -Z switch tells the compiler to use Z80 opcodes. This switch is the default mode.

-S The -S switch forces the compiler to write long quoted strings (i.e., more than 4 characters) to the binary file as they are encountered. This allows large programs with many quoted strings to compile in less memory, since without the -S switch the compiler stores quoted strings in memory in a table during compilation. Then, if an identical string appears later in the program, the compiler will match it with the string in memory and will know that the string needs to be stored only once in the /REL file. With -S set, however, no strings are stored in memory during compilation. Therefore, more memory is left for other storage needed by the compiler so slightly larger programs can be compiled. You should note, however, that the /REL file and ultimately the /CHN file may be larger since duplicate quoted strings may be contained in them.

-4 This switch is the default mode. The -4 switch allows the compiler to use the lexical conventions of Microsoft 4.51 Level II and Disk BASIC interpreters. That is, spaces are insignificant, variables with embedded reserved words are illegal, variable names are restricted to two significant characters, etc. This feature is useful if you wish to compile a source program that was coded without spaces, and contains lines such as

```
FORI=ATOBSTEPC
```

Without the -4 switch, the compiler would assign the variable "ATOBSTEPC" to the variable "FORI". With the -4 switch, it would recognize it as a FOR statement.

-5 The -5 switch tells the compiler to use BASIC 5.1 conventions. This switch clears the effect of the -4 and -T switches. If Disk BASIC lexical conventions and BASIC 5.1 execution conventions are desired, use -5-4. If BASIC lexical conventions and Disk BASIC execution conventions are desired, use -5-T. The -5 switch must be given first for these combinations to function as described.

-C The -C switch tells the compiler to relax line numbering constraints. When -C is specified, line numbers may be in any order, or they may be eliminated entirely. Lines are compiled normally, but of course cannot be targets for GOTOs, GOSUBs, etc. Be aware that while -C is set, the underline character causes the remainder of a physical line to be ignored. Also, -C causes the underline character to act as a line feed so that the next physical line becomes a continuation of the current logical line. NOTE: -C and -4 may not be used together. (Remember, -4 is the default mode.) You must specify the -5 switch whenever you want to use the -C switch.

-T This switch is the default mode. The -T switch tells the compiler to use Disk BASIC execution conventions in the following cases:

1. FOR/NEXT loops are always executed at least one time.

2. RND, TAB, SPC, POS, and LPOS perform according to Disk BASIC conventions

3. Automatic floating point to integer conversions use truncation instead of rounding except in the case where a floating point number is being converted to an integer in an INPUT statement.

4. The INPUT statement leaves the variables in the input list unchanged if you only press <ENTER>. If a "?Redo from start" message appears, then a valid input list must be entered. Pressing <ENTER> after receiving a "?Redo from start" message will generate another "?Redo from start" message.

### 3.2.4 BASIC Compiler Error Messages

The following errors may occur while a program is compiling. The BASIC Compiler outputs the two-character code for the error, along with an arrow. The arrow indicates where in the line the error occurred. For example:

```
001E 000B      20  COMMON
                   ↑ SI
```

In those cases where the compiler has read ahead before it discovered the error, the arrow points a few characters beyond the error, or at the end of the line.

As with all continuous listings on your TRS-80 screen, the error codes and messages will be displayed much too fast to read. As long as there is no more than one screenful of errors to display, you will be able to see the codes and messages clearly. If there is more than one screenful of codes and messages, you should printout the listing file so that you can see the errors clearly.

If the BASIC Compiler informs you of any fatal errors, return to Disk BASIC interpreter, load your source program, and edit the bugs. We recommend that you consult the operational, language, and feature differences discussed in the section on Editing to be sure that you have observed the

differences. A /REL file with even one fatal error will be seriously flawed. It is doubtful that such a /REL file will execute at all even if it can be loaded and linked.

Warning errors are simply indicators to remind the programmer, for example, that an array was not dimensioned and will consequently be assigned the default of 10 elements. Warning errors do not cause compiling problems but may cause runtime problems if, for example, the 11th element of an undimensioned array is referenced.

REMEMBER: Setting the -N switch allows you to print out the listing with all errors shown but much paper saved.

The error codes are as follows:

FATAL ERRORS

| CODE | ERROR |
|---|---|
| SN | Syntax Error. Caused by one of the following:<br>Illegal argument name<br>Illegal assignment target<br>Illegal constant format<br>Illegal debug request<br>Illegal DEFxxx character specification<br>Illegal expression syntax<br>Illegal function argument list<br>Illegal function name<br>Illegal function formal parameter<br>Illegal separator<br>Illegal format for statement number<br>Illegal subroutine syntax<br>Invalid character<br>Missing AS<br>Missing equal sign<br>Missing GOTO or GOSUB<br>Missing comma<br>Missing INPUT<br>Missing line number<br>Missing left parenthesis<br>Missing minus sign<br>Missing operand in expression<br>Missing right parenthesis<br>Missing semicolon<br>Name too long<br>Expected GOTO or GOSUB<br>String assignment required<br>String expression required<br>String varible required here<br>Illegal syntax |

Variable required here
Wrong number of arguments
Formal parameters must be unique
Single variable only allowed
Missing TO
Illegal FOR loop index variable
Missing THEN
Missing BASE
Illegal subroutine name

OM Out of Memory
Array too big
Data memory overflow
Too many statement numbers
Program memory overflow

SQ Sequence Error
Duplicate statement number
Statement out of sequence

TM Type Mismatch
Data type conflict
Variables must be of same type

TC Too Complex
Expression too complex
Too many arguments in function call
Too many dimensions
Too many variables for LINE INPUT
Too many variables for INPUT

BS Bad Subscript
Illegal dimension value
Wrong number of subscripts

LL Line Too Long

UC Unrecognizable Command
Statement unrecognizable
Command not implemented

OV Math Overflow

/0 Division by Zero

DD Array Already Dimensioned

FN FOR/NEXT Error
FOR loop index variable already in use
FOR without NEXT
NEXT without FOR

FD Function Already Defined

UF Function Not Defined

WE WHILE/WEND Error
WHILE without WEND
WEND without WHILE

/E Missing "-E" Switch

/X Missing "-X" Switch

LS String constant too long

IN %INCLUDE error

WARNING ERRORS

ND Array Not Dimensioned

SI Statement Ignored
Statement ignored
Unimplemented command

## 3.3 LINK-80 LINKING LOADER

A linking loader performs two important programming functions. First, it loads into memory one or more program files the programmer selects. The files that LINK-80 loads are called REL files. REL files are created during the compiling process and contain relocatable machine code. So, a REL file is not an executable file. Converting a REL file into an executable object file, a process known as linking, is the second function of the linking loader. Specifically, the linking loader (L80) searches through the /REL file (or files if more than one has been loaded) and finds all references to subroutines needed to perform BASIC or other functions; such as floating point addition, printing data to the screen, and so on. These are called Global References. Since they are not yet in memory, they are referred to as Undefined Globals.

Some of the subroutines needed to satisfy these Undefined Globals are in BRUN, the runtime module which will be brought into memory just prior to execution. Others, the less commonly used subroutines, are in BASLIB, the BASIC subroutine library. For each BASIC function, there is either a complete subroutine (or series of subroutines) stored in BASLIB or there is a reference to a subroutine stored in BRUN.

L80 searches BASLIB to satisfy Undefined Globals. If the subroutine needed is stored in BASLIB, L80 "links" that subroutine to the loaded program(s). If the required subroutine is stored in BRUN, L80 sets up the code necessary for the program to find the subroutine in BRUN.

The final action of the linking loader, if the programmer requests it, is to save the loaded program(s) and the linked routines in a single executable disk file. This file will be given the extension /CHN automatically, unless the programmer specifies otherwise. (/CHN is used as the default extension to distinguish these files from /CMD files which do not need BRUN to execute.)

In addition to these basic link loading functions, LINK-80 can:

1. Load and link assembly language routines written with MACRO-80, Microsoft's Assembly Language Development System.

2. Load and link FORTRAN subroutines created with the Microsoft FORTRAN-80 compiler, plus search the FORTRAN library, FORLIB.

3. Allow the programmer control over where program and data areas are to be placed.

So, LINK-80 allows the programmer to link parts of a user library of FORTRAN-80 or MACRO-80 subroutines into a BASIC compiled program. In this way, LINK-80 is more than a necessary tool for using BASIC Compiler; LINK-80 gives the BASIC programmer the means to extend the power of BASIC programs quickly without recoding FORTRAN or assembly language routines.

NOTE

FORTRAN and ALDS are separate products that are available from Microsoft Consumer Products.

### 3.3.1 LINK-80 Command Line

This section is designed to give you an understanding of LINK-80 and how the command lines work.

Just as the BASIC Compiler command line has several fields which may have more than one part, LINK-80 command lines also have several fields with several possible parts. In command line format, the fields are:

```
RELfile[,RELfile,...][,CHNfilename-N][-switch...]-E
```

Let's discuss each field.

REL FILE FIELD (filename)

At least one entry is required in this field. The filenames you enter in this field are the names of the /REL files you want to link and load.

If you do not enter a filename in this field, you will receive the error message

?NOTHING LOADED

As the square brackets and ellipses suggest, you may enter as many filennames in this field as you choose, as long as

1. the command line does not exceed one physical line
2. you separate compiled filenames with commas.
3. the files you list are /REL compiled files. (The compiled file may have any filename extension you gave during the compiling process, but the extension must be included here when you name the file. LINK-80 assumes that the file will have the default extension /REL.)
4. No more than one of the /REL files is a compiled BASIC program. Compiled BASIC programs cannot be linked together.

/CHN FILE FIELD (,filename-N)

The /CHN filename field is optional. However, if you want to save the program on disk, which is the goal of compiling anyway, you must specify a /CHN filename followed by the -N switch. (The -N switch tells LINK-80 to save the linked file on disk under the flename the programmer specifies.)

If you list the -N switch without the /CHN filename, you will receive the message

?NOTHING LOADED

If you list the /CHN filename without the -N switch, no /CHN file will be saved on disk.

SWITCH (-Switch)

One entry is required in this field and the others are optional.

The required entry is the -E switch.

Before we discuss specific switches, let's review switches generally. A switch is a command to LINK-80 (in this case) to perform an additional or alternate function. The device called the Switch acts to "switch on" the function only when the switch letter is given in the command line. All switches must be preceded by a dash (-). In command lines, it is permissable to list more than one switch, as long as each switch is preceded by a dash (-).

Unlike BASCOM command line switches, however, LINK-80 command line switches are not always placed at the end of the command line. Most are placed at the end of the command line, but some must be placed at the beginning, and some in the middle.

Now, let's look at the switches available under LINK-80. REMEMBER: Do not confuse these switches with the BASIC Compiler switches.

The chart below summarizes the switch functions. Full descriptions of the switches follow the chart.

| SWITCH | ACTION |
|---|---|
| -E | Exit to TRSDOS |
| -N | Save all previously loaded programs and subroutines using the name immediately preceding -N |
| -N:P | Alternate form of -N; save only program area |
| -O | Octal radix |
| -H | Reset to hexadecimal radix (default) |
| -S | Search the library named immediately preceding -S |
| -R | Reset LINK-80 and default start address to 5200H and default extension for saved files to /CMD. |
| -P | Set start address for programs and data. If used with -D, -P sets only the program start. |
| -D | Set start address for data area only. |
| -U | List undefined globals and program and data area information (a direct command) |
| -M | List complete global reference map |

Two switches will be used in every linking session.

SWITCH ACTION

-N This switch saves a memory image of the executable file on disk, using the filename and extension you specify.. If you do not specify an extension, the default extension for the saved file is /CHN. Unless this switch is given in the command line, no memory image of the linked file is saved on disk. To specify which drive contains the diskette for saving the memory image, insert the drive number (:d) between the filename and -N.

Once saved on disk, you need only type BRUN filename at TRSDOS command level to run the program. The -N switch must immediately follow the filename of each file you wish to save, and it does not take effect unless a -E switch is given following it.

You will use this switch almost everytime you link a /REL file.

The default condition of saving a /CHN file is to save both program and data areas. If you wish to save only the program to make your disk files smaller, use the -N switch in the form -N:P. With this switch set, only the program code will be saved. Do not use this -N:P feature if you compiled your program with the -S switch.

-E The -E switch causes LINK-80 to execute then causes program control to Exit from LINK-80 and return to TRSDOS command level. Every link loading command line should end with the -E switch. While it is possible to exit LINK-80 in other ways (press <BREAK> when at LINK-80 command level), the -N switch has no effect until LINK-80 sees the -E switch. So, the -E switch is essentially required for a successful link loading session.

These two switches are all that are required in most LINK-80 operations. Some additional functions are available through the use of other switches which allow programmers to manipulate the LINK-80 processes in more detail.

SWITCH ACTION

-S The -S switch causes LINK-80 to search the file named immediately prior to the switch for routines, subroutines, definitions for globals, and so on. In a command line, the filename with the -S switch appended must be separated from the rest of the command line by commas.

The -S switch is used to search library files only, such as BASLIB or FORLIB.

You rarely need to give the -S switch. Only under the following conditions is it required:

1. Use BASLIB-S if you have only one drive (see steps for Running LINK-80 given above).

2. Use FORLIB-S to search the FORTRAN runtime library if one or more of the programs you are link loading is a FORTRAN program.

-R The -R switch "resets" LINK-80 to its initialized condition. LINK-80 scans the command line before it begins the functions commanded. As soon as LINK-80 sees the -R switch, all files loaded are ignored, LINK-80 resets itself, and the asterisk (*) prompt is returned showing that LINK-80 is running and waiting for you to enter a command line.

The version of LINK-80 supplied with BASIC Compiler defaults the initial load address to 8A00H. Using the -R switch resets the default load address to 5200H. The default save file extension is /CHN. Using the -R switch changes the default extension to /CMD.

If you still want to load BASIC programs, you must explicitly specify the proper load address (8A00H) and save filename extension (/CHN), or you must restart LINK-80 from TRSDOS.

The -R switch is useful when you want to load files other than a BASIC program.

-P The -P switch is used to set both the program and data origin. The format of the -P switch is

-P:<address>

The address value must be expressed in the current radix. The default radix is hexadecimal. You

will know if the radix is set for a base other than hexadecimal because the radix can only be changed by giving a switch in the LINK-80 command line.

The default value for the -P switch is :8A00, set for the current version of BRUN which occupies the addresses 5200-8A00. If you want to link programs other than BASIC programs, set the -P switch to :5200, or use the -R switch (see -R switch description above).

REMEMBER: The -P switch takes effect as soon as it is seen, but it does not affect files already loaded. So be sure to place the -P switch before any files you want to load at the specified address. The -P switch and -D switch, when used, must be separated from the RELfilename by a comma. For example,

```
L80 -P:8A00,GRAF,GRAF-N-E
```

This switch may be used to set program and data origin higher to make room for small machine language routines, as described in the section on the USRn function.

-D The -D switch sets the data area origin by itself. Since the program origin always starts exactly at the end of the data area, unless otherwise specified, the -D switch used by itself has the exact same effect as the -P switch used by itself. The format for the -D switch is the same as the -P switch format:

```
-D:<address>
```

The address for the -D switch must be in the current radix. (Hexadecimal is the default radix.)

When the -P switch is used with the -D switch, Data areas load starting at the address given with the -D switch. (The program will be loaded beginning at the program origin given with the -P switch.) This is the only occassion when the address given in -P: is the start address for the actual program code.

The -D switch, like the -P switch, takes effect as soon as LINK-80 "sees" the switch (the effect is not deferred until linking is finished), but the

-D switch has no effect on programs already loaded. So it is important to place the -D switch (as well as the -P switch) before the data (and programs) you want to load at the address specified.
The -P switch and -D switch, when used, must be separated from the RELfilename by a comma. For example,

```
L80 -P:8A00,GRAF,GRAF-N-E
```

ADDITIONAL NOTE FOR -P AND -D SWITCHES

If your program is too large for the loader, you will sometimes be able to load it anyway if you use -D and -P together. This way you will be able to load programs and data up to a combined total of 30K. While LINK-80 is loading and linking, it builds a table consisting of five bytes for each program relative reference. If you use the -D, -E, or -X switch, this table contains at least five bytes for every line number. By setting both -D and -P, you eliminate the need for LINK-80 to build this table, thus giving you some extra memory to work with.

To set the two switches, look to the end of the LST file listing. Take the number for the total of data, add that number to 8A00H, add another 100H+1, and the result should be the -P: address for the start of the program area. The -D switch should be set to -D:8A00.

-O The -O switch sets the current radix to Octal. If you have a reason to use octal values in your program, give the -O switch in the command line. If you can think of no reason to switch to octal radix, then there is no reason to use this switch.

-H The -H switch resets the current radix to Hexadecimal. Hexadecimal is the default radix. You do not need to give this switch in the command line unless you previously gave the -O switch and now want to return to hexadecimal.

-U The -U switch tells LINK-80 to list all undefined globals, to the point in the link session when LINK-80 encounters the -U switch.. While this switch is not a default setting, if your program contains any undefined globals, they will be listed automatically, just as if you had set the -U switch. If your program contains no undefined globals, the actions controlled by the -U switch will not occur

unless -U is given in a command line that does not end with a -E switch. Globals are the names of assembly language subroutines that are called from the REL file. If LINK-80 cannot find the routine, the global is undefined. Unless you have written some of your own subroutines and have directed LINK-80 to load and link them with your compiled program, you should have no need to use this switch. BASLIB provides definitions for the globals you need to run your program.

In addition to listing undefined globals, the -U switch directs LINK-80 to list the origin and the end of the program and data areas. However, the program portion of the information is listed only if the -D switch was also given. If the -D switch was not given also, the program is stored in the data area, and the origin and end of the data area include the origin and end of the program.

-M The -M switch lists all globals, both defined and undefined, on the screen. The listing cannot be sent to a printer. In the listing, defined globals are followed by their values, and undefined globals are followed by an asterisk (*).

Both the -M switch and the -U switch list the program and data area information.

### 3.3.2 LINK-80 Error Messages

**?Loading Error** — The last file given for input was not a properly formatted LINK-80 object file.

**?Out of Memory** — Not enough memory to load program.

**?Command Error** — Unrecognizable LINK-80 command.

**?<file> Not Found** — <file>, as given in the command string, did not exist.

**%2nd COMMON Larger /XXXXXX/**
You will not receive this error because the COMMON statement is not implemented.

**%Mult. Def. Global YYYYYY**
More than one definition for the global (internal) symbol YYYYYY was encountered during the loading process. This means two subroutines with the same ENTRY point were specified in the LINK-80 command line.

**%Overlaying { Program / Data } Area**

**?Intersecting { Program / Data } Area**
If you receive either of these error messages, you have set the -D and -P switches too close together. Reset the :<address> portion of both switches so that the locations are farther apart.

**Origin { Above / Below } Loader Memory, Move Anyway (Y or N)?**
Loader memory is 5200H to high memory. If you received this error message, you specified the -D or the -P switch with an address outside this range. Reset the :<address> portion of the switch(es).

**?Can't Save Object File**
A disk error occurred when the file was being saved. Almost always when you receive this message, you can assume that there is not enough disk space free in which to store the program.

## 3.4 BRUN/BASLIB

One of the major benefits of a compiled program is that you can run the program very easily from TRSDOS command level. It is not necessary to enter BASIC, then load the program, then run it.

However, your /CHN file requires that runtime routines be available to answer the CALLs compiled into the /CHN file. The runtime routines reside on BASIC Compiler diskette #2.

The runtime routines are divided into two parts. One part is the BRUN runtime module, which contains the most commonly used of these runtime routines. Each time the programmer runs a program, BRUN must be in memory. The subroutines are loaded as a fixed module into the same memory space every time-- addresses 5200H-8A00H. The remainder of the runtime routines needed to run programs on your TRS-80 are contained in a relocatable file called BASLIB/REL. These routines are brought into memory and become a part of your program, but only if the specific routine is necessary (i.e., used in your program).

To run a CHN file (a program that has been compiled and link loaded) from TRSDOS command level, first arrange your diskettes so that both the BRUN runtime module and your /CHN file are available to your TRS-80.

## ONE-DRIVE SYSTEMS

The simplest method for running your program is to simply enter

```
BRUN
```

When BRUN returns the asterisk (*) prompt, enter

<filename> (without the /CHN extension)

Your program should now run as usual, but much faster.

A second method is available, although it is not as "neat." BACKUP BASCOM diskette #2, then delete the BASLIB/REL file from the backup diskette. Then save your CHN file on this diskette.

Once you have your program object file and BRUN on the same diskette, simply enter BRUN <filename> (without the /CHN extension). This method is the same as for multi-drive systems.

## MULTI-DRIVE SYSTEMS

Place BASCOM diskette #2 in one of the drives and your TRSDOS program diskette or formatted non-TRSDOS program diskette in the other drive. (If you just finished link loading, you do not need to change any diskettes, but may proceed.)

Now simply enter

```
BRUN <filename>
```

Also, with a multi-drive system, you can use the TRSDOS COPY command to copy BRUN onto your program diskette, if you wish.

## RUNTIME ERROR MESSAGES

Runtime error messages are listed on your monitor when they occur. Your TRS-80 will then automatically reboot TRSDOS.

The BASIC Compiler runtime system prints error messages followed by an address. For example,

```
OVERFLOW AT ADDRESS 8A8E
```

When you receive a runtime error which gives the address of the error, you need to review the listing file.

Subtract the hexadecimal address of the program origin from the error address. Find the difference in the left column of hexadecimal numbers in the list file. When you find the line which corresponds to the difference between the program origin and the error address, the line number will be the line which contains the error.

NOTE

If you deleted the listing file, you will need to recompile it. Enter the command:

```
BASCOM ,LSTfile:d=BASfile
```

Or, if you have a printer available, it is better to enter:

```
BASCOM ,*PR=BASfile
```

This option will give you hardcopy to work with, plus no disk file will be created, saving disk space.

The error codes and messages are given here for those programmers who wish to implement the ON ERROR GOTO error trapping routine. In this case, the error codes are used in the error trapping routine to identify the errors. Refer to the BASIC-80 Reference Manual for a description of ON ERROR GOTO and RESUME statements.

The error numbers listed correspond to the value returned by ERR for BASIC version 5 (refer to the compiler -5 switch). If you want to compute the ERR number returned by Disk BASIC, multiply the listed error number by 2 then subtract 2.

| NUMBER | MESSAGE |
|---|---|
| 2 | Syntax error<br>A line is encountered that contains an incorrect sequence of characters in a DATA statement. |
| 3 | RETURN without GOSUB<br>A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement |

4 Out of data
A READ statement is executed when there are no DATA statements with unread data remaining in the program.

5 Illegal function call
A parameter that is out of range is passed to a math or string function. An FC error may also occur as the result of:

1. a negative or unreasonably large subscript
2. a negative or zero argument with LOG
3. a negative argument to SQR
4. a negative mantissa with a non-integer exponent
5. a call to a USR function for which the starting address has not yet been given
6. an improper argument to ASC, CHR$, MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, or ON...GOTO
7. a string concatenation that is longer than 255 characters

6 Floating overflow or integer overflow
The result of a calculation is too large to be represented in BASIC-80's number format. If underflow occurs, the result is zero and execution continues without an error.

9 Subscript out of range
An array element is referenced with a subscript that is outside the dimensions of the array.

11 Division by zero
A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power. Machine infinity with the sign of the numerator is supplied as the result of the division, or positive machine infinity is supplied as the result of the involution, and execution continues.

14 Out of string space
String variables exceed the allocated amount of string space.

20 RESUME without error
A RESUME statement is encountered before an error trapping routine is entered.

21 Unprintable error
An error message is not available for the error condition which exists. This is usually caused by an ERROR with an undefined error code.

50 Field overflow
A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random file.

51 Internal error
An internal malfunction has occurred in Disk BASIC-80. Report to Microsoft the conditions under which the message appeared.

52 Bad file number
A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified at initialization.

53 File not found
A RUN, CHAIN, KILL, or OPEN statement references a file that does not exist on the current disk.

54 Bad file mode
An attempt is made to use PUT, GET, or LOF with a sequential or to execute an OPEN with a file mode other than I, O, R, or D.

55 File already open
A sequential output mode OPEN is issued for a file that is already open; or a KILL is given for a file that is open.

57 Disk I/O error
An I/O error occurred on a disk I/O operation. It is a fatal error, i.e., the operating system cannot recover from the error.

58 File already exists
The filename specified is identical to a filename already in use on the disk.

61 Disk full
All disk storage space is in use.

62 Input past end
An INPUT statement is executed after all the data in the file has been INPUT, or for a null (empty) file. To avoid this error, use the EOF function to detect the end of file.

63 Bad record number
In a PUT or GET statement, the record number is either greater than the maximum allowed (32767) or equal to zero.

64 Bad file name
An illegal form is used for the filename with RUN, CHAIN, KILL, or OPEN (e.g., a filename with too many characters).

67 Too many files
An attempt is made to create a new file (using OPEN) when the directory is full.

# APPENDIX A

## FORMAT OF LINK COMPATIBLE OBJECT FILES

### NOTE

This section is reference material for users who wish to know the load format of LINK-80 relocatable object files. This section does not contain material necessary to the operation of LINK-80.

LINK-compatible object files consist of a bit stream. Individual fields within the bit stream are not aligned on byte boundaries, except as noted below. Use of a bit stream for relocatable object files keeps the size of object files to a minimum, thereby decreasing the number of disk reads/writes.

There are two basic types of load items: Absolute and Relocatable. The first bit of an item indicates one of these two types. If the first bit is a 0, the following 8 bits are loaded as an absolute byte. If the first bit is a 1, the next 2 bits are used to indicate one of four types of relocatable items:

| | |
|---|---|
| 00 | Special LINK item (see below). |
| 01 | Program Relative. Load the following 16 bits after adding the current Program base. |
| 10 | Data Relative. Load the following 16 bits after adding the current Data base. |
| 11 | Common Relative. Load the following 16 bits after adding the current Common base. |

Special LINK items consist of the bit stream 100 followed by:

a four-bit control field

an optional A field consisting of a two-bit address type that is the same as the two-bit field above except 00 specifies absolute address

an optional B field consisting of 3 bits that give a symbol length and up to 8 bits for each character of the symbol

A general representation of a special LINK item is:

1 00 xxxx yy nn zzz + characters of symbol name

A field B field

xxxx Four-bit control field (0-15 below)
yy Two-bit address type field
nn Sixteen-bit value
zzz Three-bit symbol length field

The following special types have a B-field only:

0 Entry symbol (name for search)
1 Select COMMON block
2 Program name
3 Request library search
4 Extension LINK items (see below)

The following special LINK items have both an A field and a B field:

5 Define COMMON size
6 Chain external (A is head of address chain, B is name of external symbol)
7 Define entry point (A is address, B is name)

The following special LINK items have an A field only:

8 External - offset. Used for JMP and CALL to externals
9 External + offset. The A value will be added to the two bytes starting at the current location counter immediately before execution.
10 Define size of Data area (A is size)
11 Set loading location counter to A
12 Chain address. A is head of chain, replace all entries in chain with current location counter. The last entry in the chain has an address field of absolute zero.
13 Define program size (A is size)
14 End program (forces to byte boundary)

The following special Link item has neither an A nor a B field:

15 End file

An Extension LINK item follows the general format of a B-field-only special LINK item, but the contents of the B-field are not a symbol name. Instead, the symbol area contains one character to identify the type of Extension LINK item, followed by from 1 to 7 characters of additional information.

Thus, every Extension LINK item has the format:

1 00 0100 zzz i jjjjjjj

where

| | |
|---|---|
| zzz | may be any three bit integer (with 000 representing 8) |
| i | is an eight bit Extension LINK item type identifier |
| jjjjjjj | are zzz-1 eight bit characters of information whose significance depends on i |

At present, there is only one Extension LINK item:

i = X'35' COBOL overlay segment sentinel

zzz = 010 (binary)

j = COBOL segment number -49 (decimal)

When the overlay segment sentinel is encountered by the linker, the current overlay segment number is set to the vaue j+49. If the previously existing segment number was non-zero and a -N switch is in effect, the data area is written to disk in a file whose name is the current program name and whose extension is Vnn, where nn are the two hexadecimal digits representing the number j+49 (decimal).

# APPENDIX B

## MEMORY MAPS

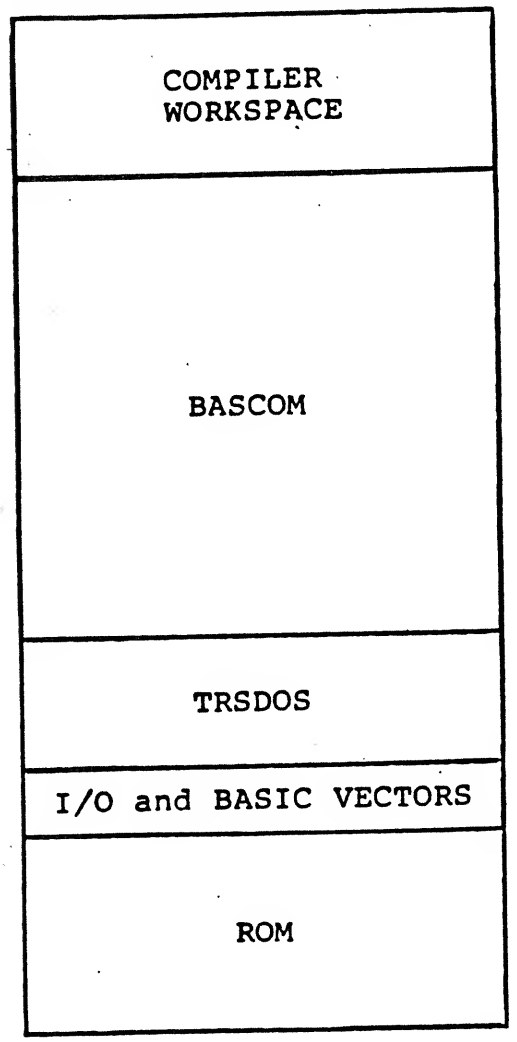
These memory maps are supplied to give you a general idea of what areas of memory are used during the BASIC Compiler processes. Unless a specific address is given, the lines indicate only a general idea of the distribution of programs and data in memory.

These memory maps should also provide you some idea of the memory areas available for the -D and -P switches.

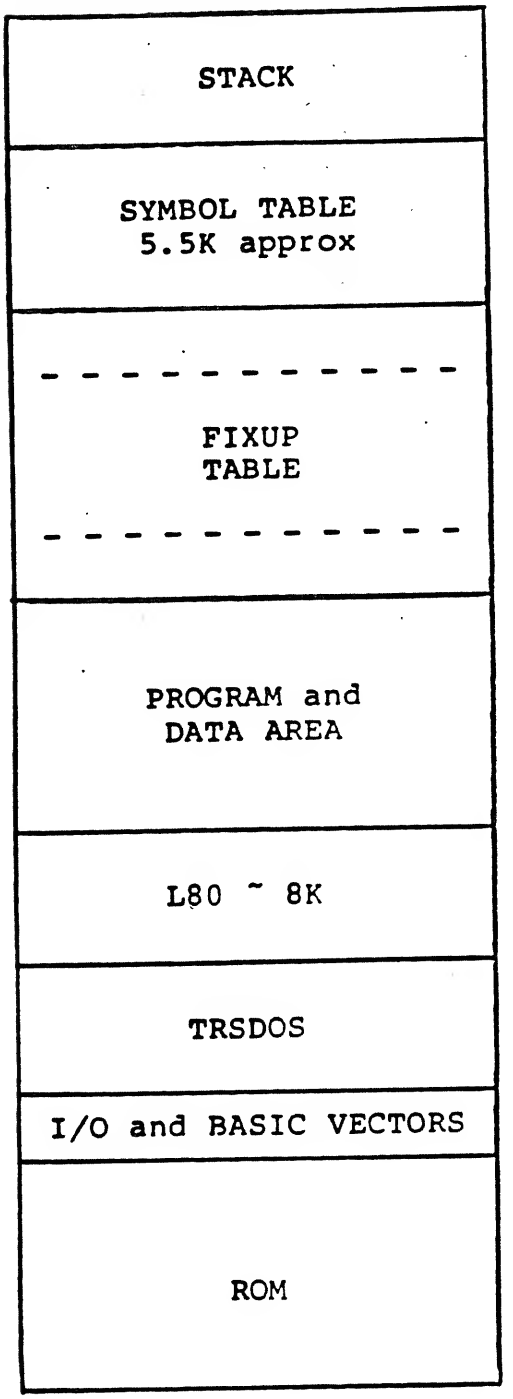
NOTE

All values are hexadecimal

DURING COMPILER OPERATION



DURING LOADER OPERATION




FFFF

varies

with -D-P, 150 bytes max
otherwise may be huge

varies

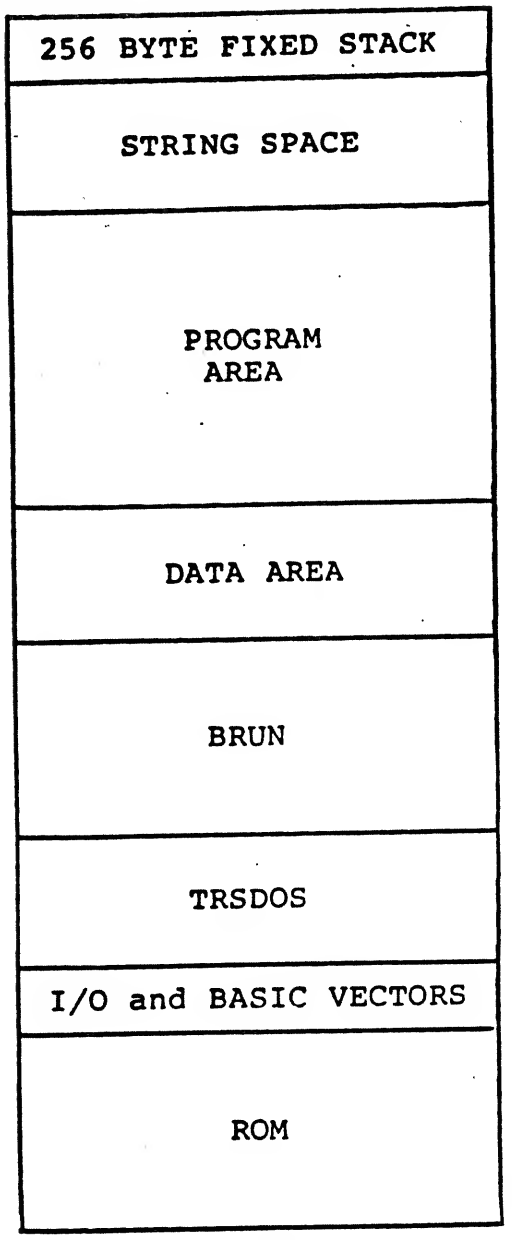
5200

4200

3000

0000

DURING PROGRAM EXECUTION



# INDEX